Deep learning applied to fundamental science
Lecture 2

mPP

hls4ml

Maurizio Pierini
CERN

CMS

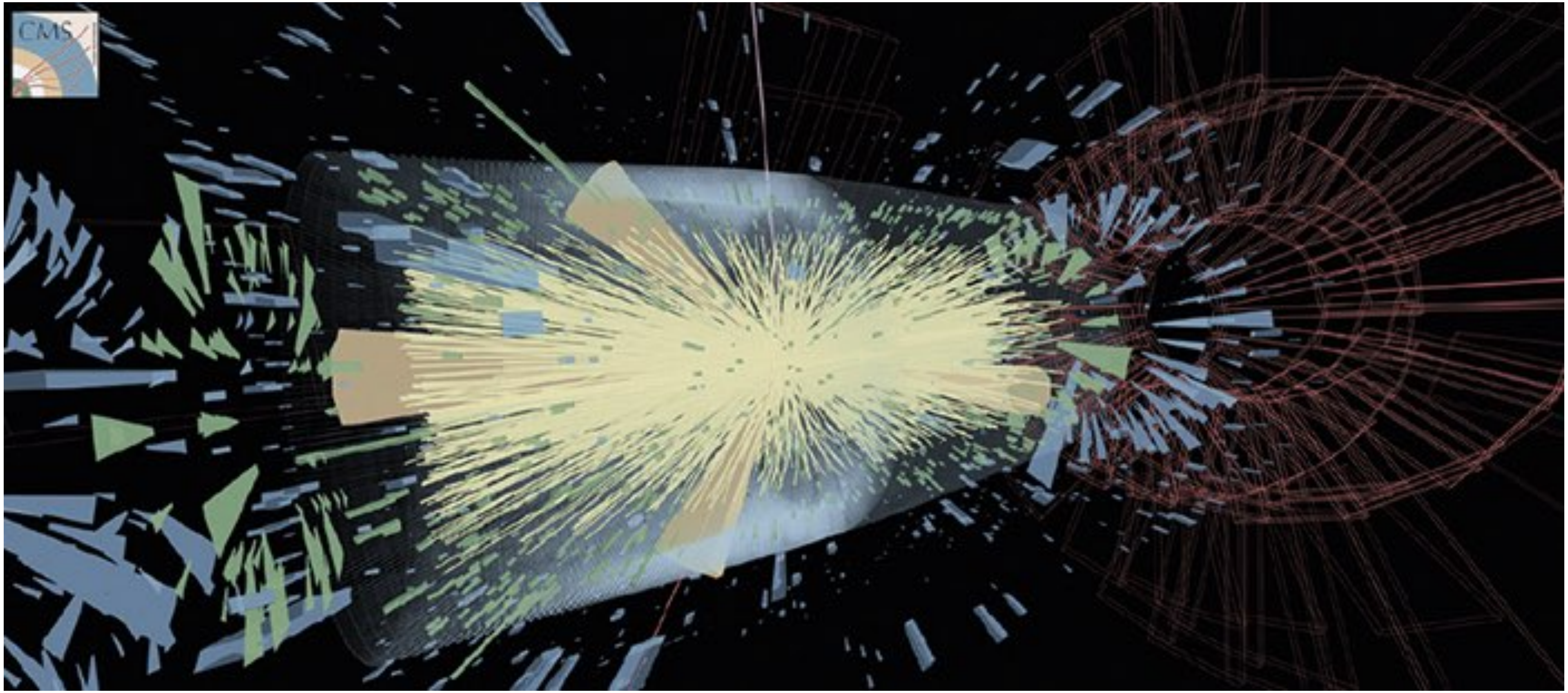
CERN

erc European Research Council

Plan for this week

Day 1	Introduction	Hands-on session	
Day 2	Applications at the LHC	Deep Learning for Discovery	Hands-on session

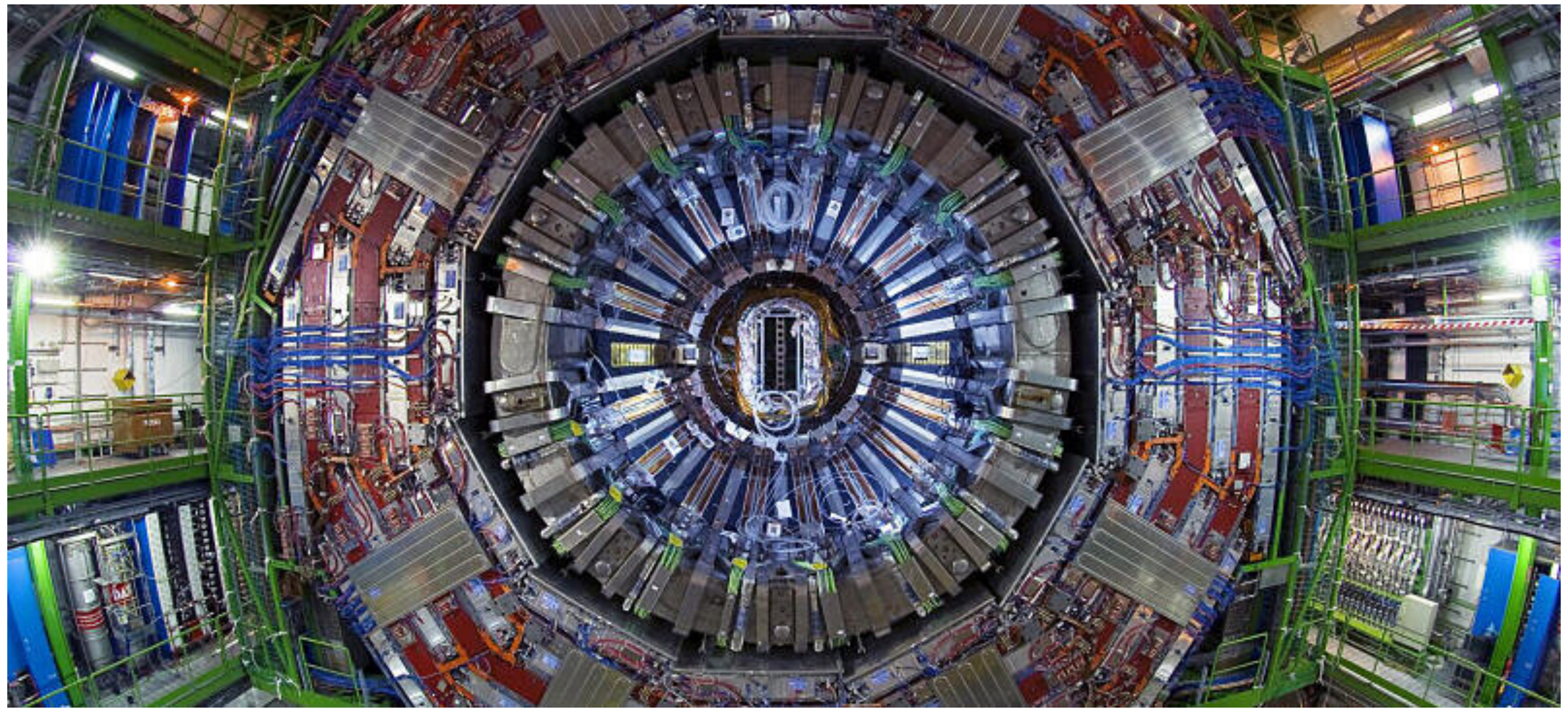
European Research Council

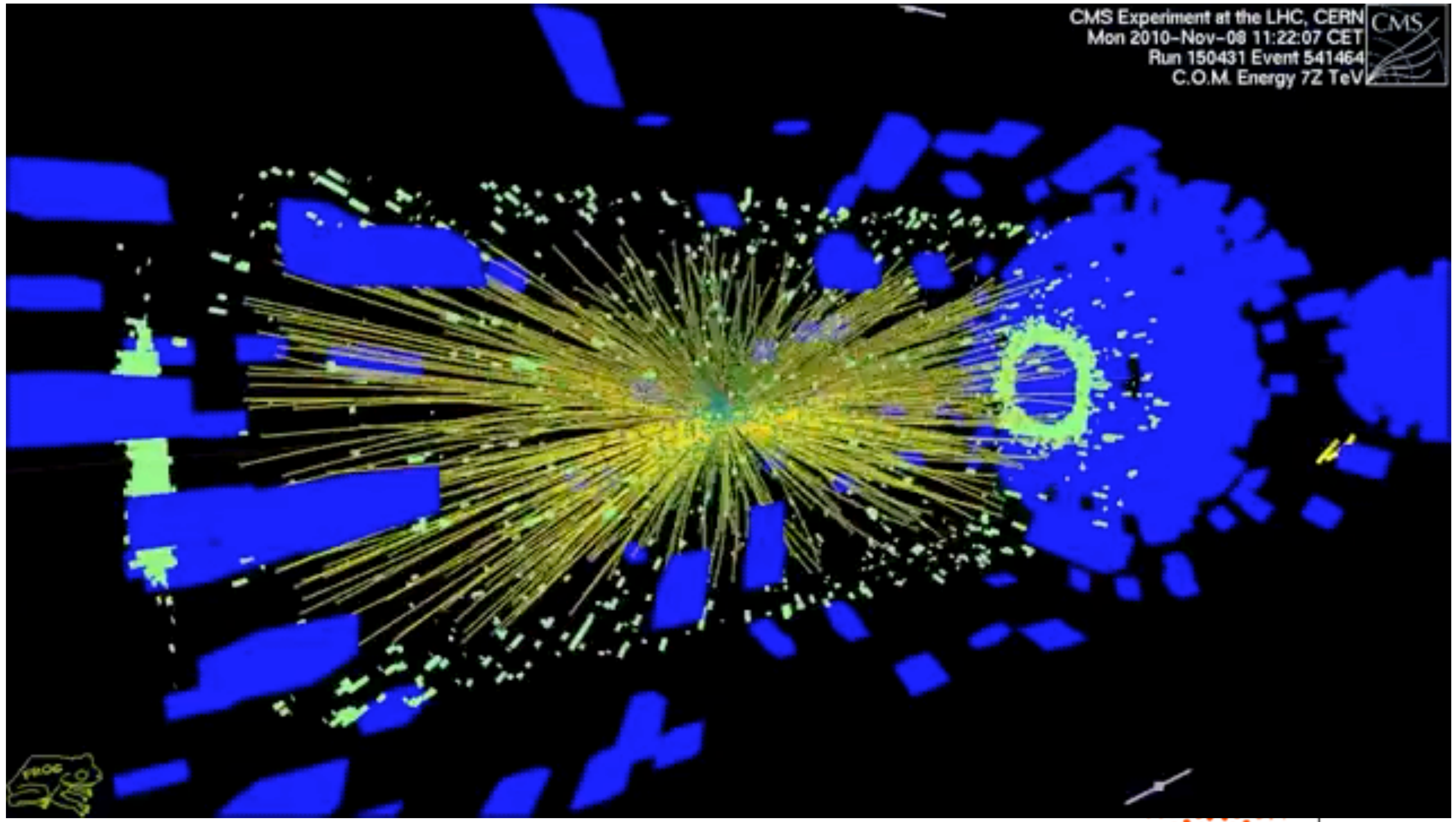


Event Reconstruction with Graph Networks

What LHC data look like

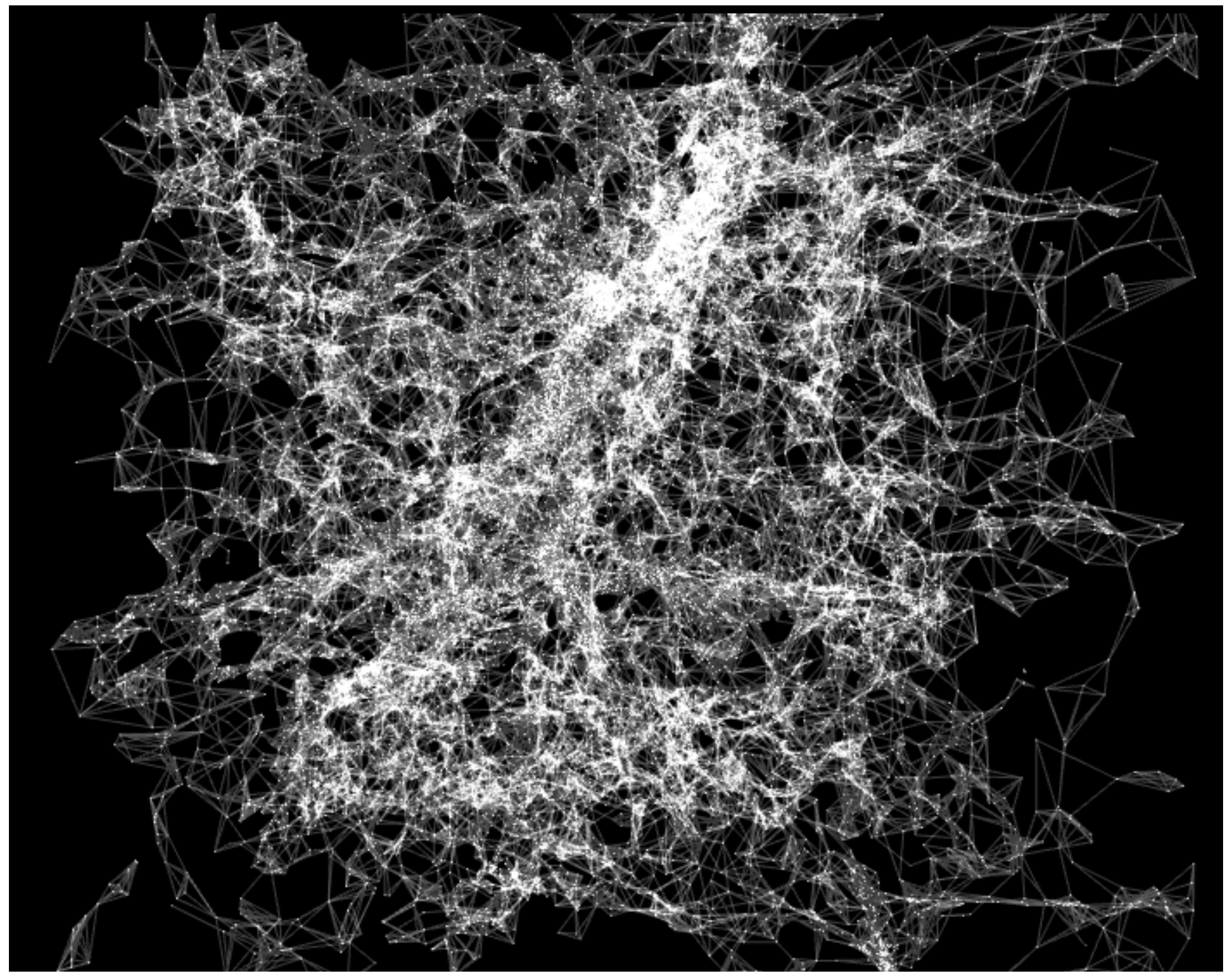
- We saw yesterday how images are processed y Convolutional Networks
- Problem: LHC data are not images:
  - difficult to fit an irregular array of sensors (unordered set of dots in some feature space) in a regular array of pixels
- One can deal with this problem loosing some information
  - pixelate the data with a coarser binning (as we did for jets)
- Or using some network that works better with sparse and irregular arrays

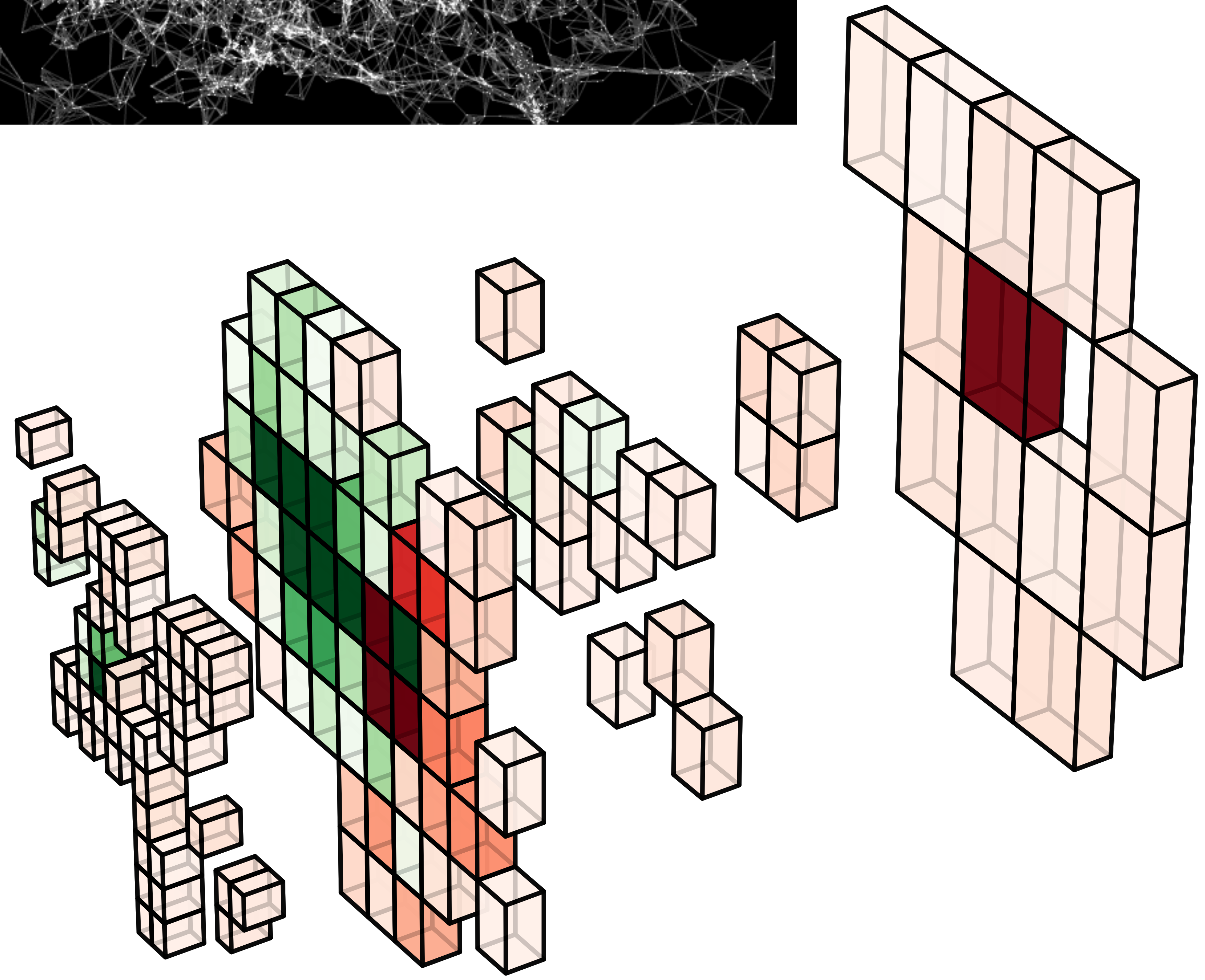




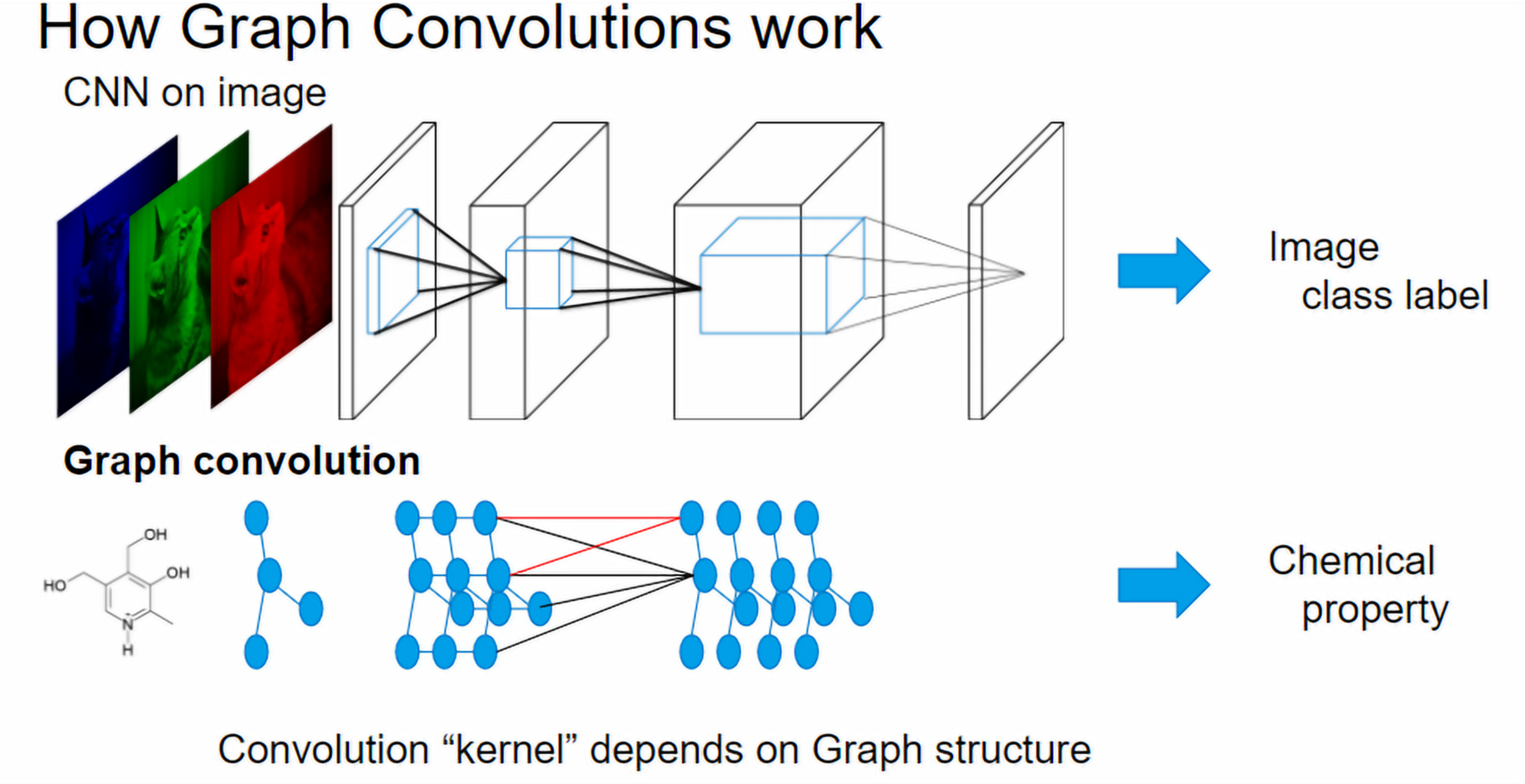
A generic problem in science

- *Many scientific problems have this issue:*
  - *Galaxies or star populations in sky*
  - *Sensors from HEP detector*
  - *Molecules in chemistry*
- *These data can all be seen as sparse sets in some abstract space*
  - *each element of the set being specified by some array of features*
  - *Some of these features (or function of) could be seen as coordinates in some random space*



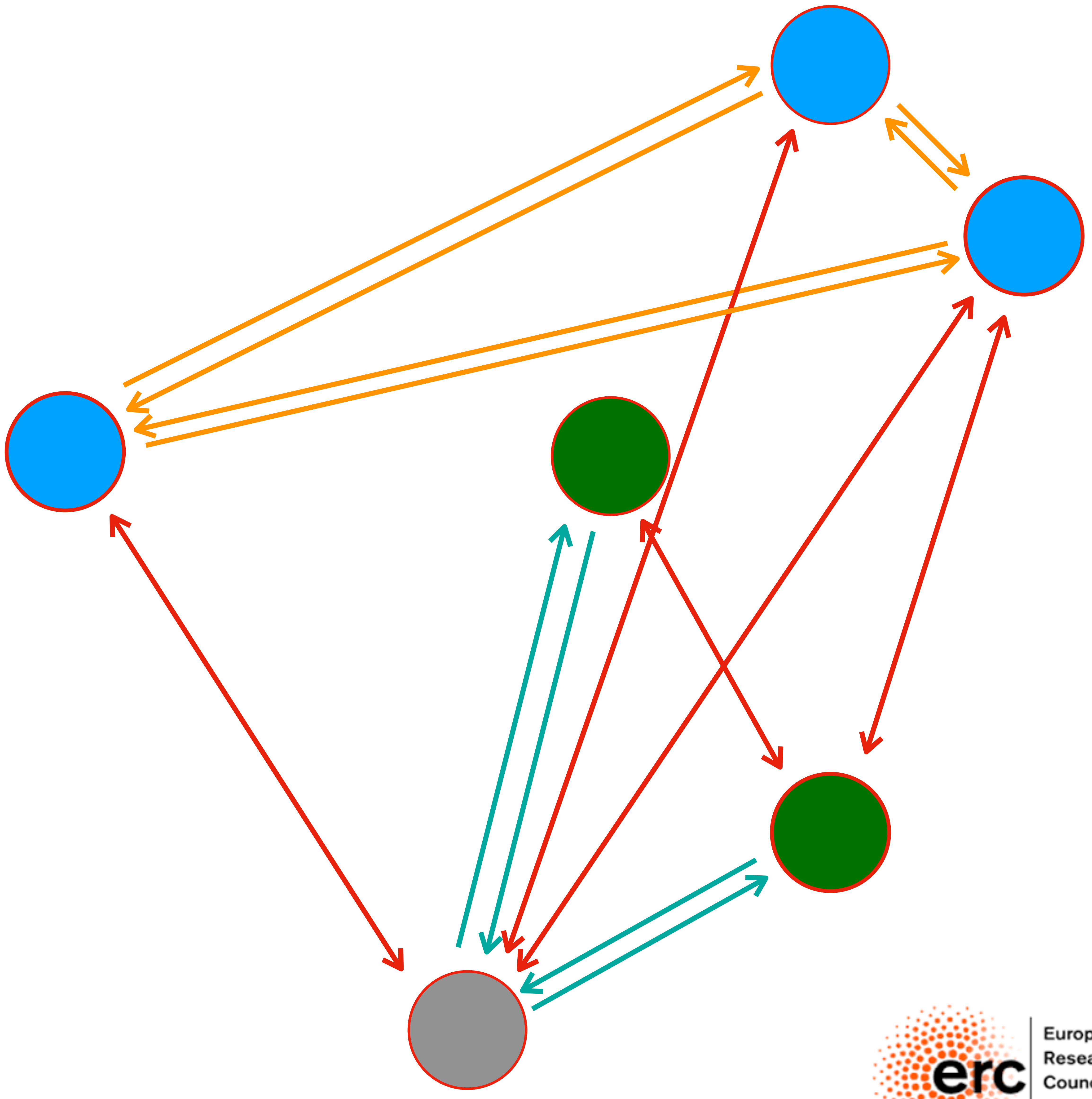


Generalising CNN to point clouds



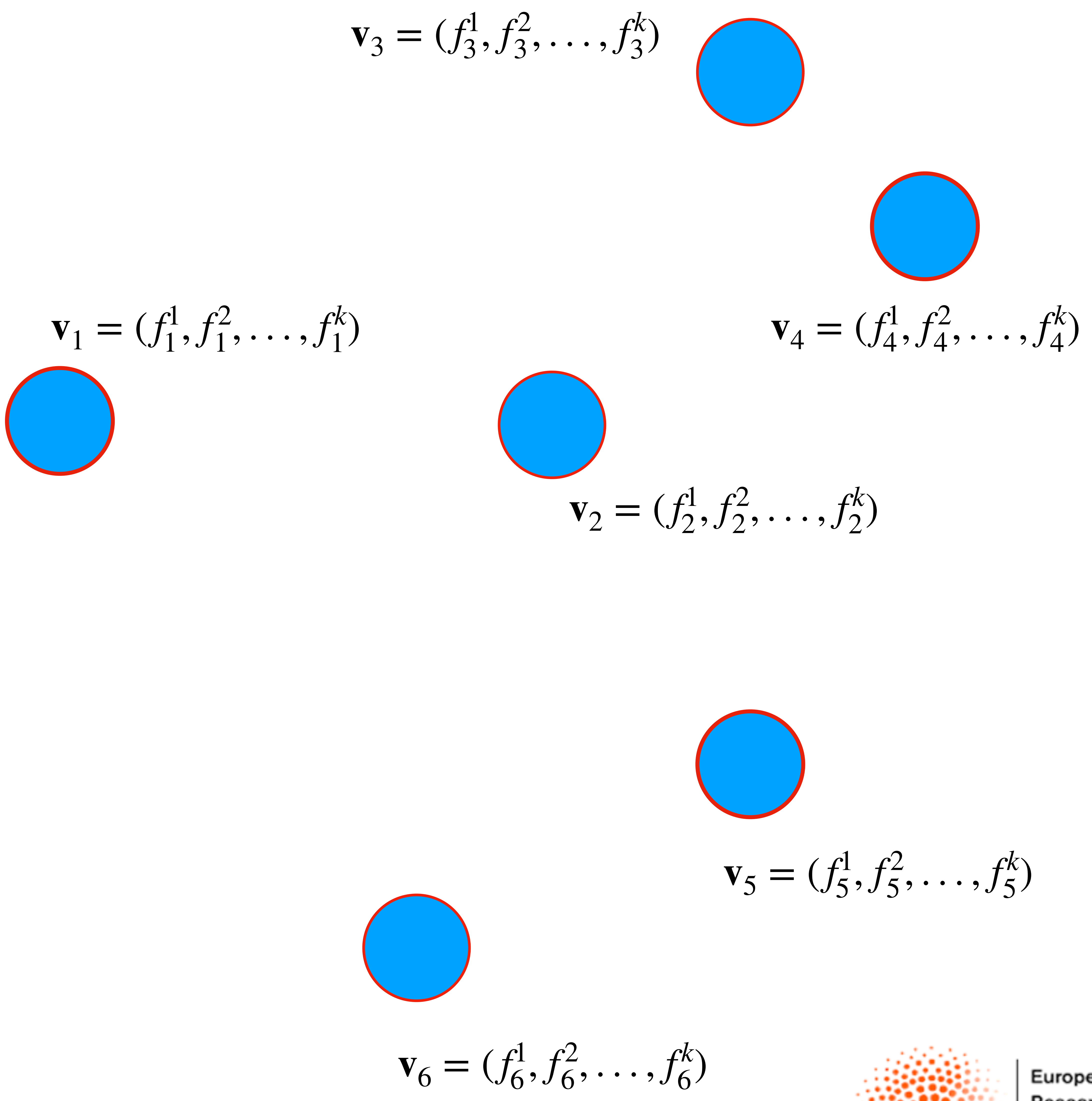
Building a Graph

- *The input is a set of vertices V connected by edges E*
  - *Edges can be directional*
  - *Graphs can be fully connected (N^2)*
  - *Or you could use some criterion (e.g., nearest k neighbours in some space) to reduce number of connections*
  - *if more than one kind of vertex, you could connect only Vs of same kind, of different kind, etc*
- *The (V,E) construction is your graph. Building it, you could enforce some structure in your data*
  - *If you have no prior, then go for a directional fully connected graph*



Graph Networks

- Once you have a graph, you want to learn from it
  - **Each item in a dataset is represented as a set of vertices (like pixels in an image)**
  - **Each vertex is represented by a vector of features (like RGB indices for images**
- Vertices are connected through links
- Messages are passed through links and aggregated on the vertices
- A new representation of each node is created, based on the information gathered across the graph

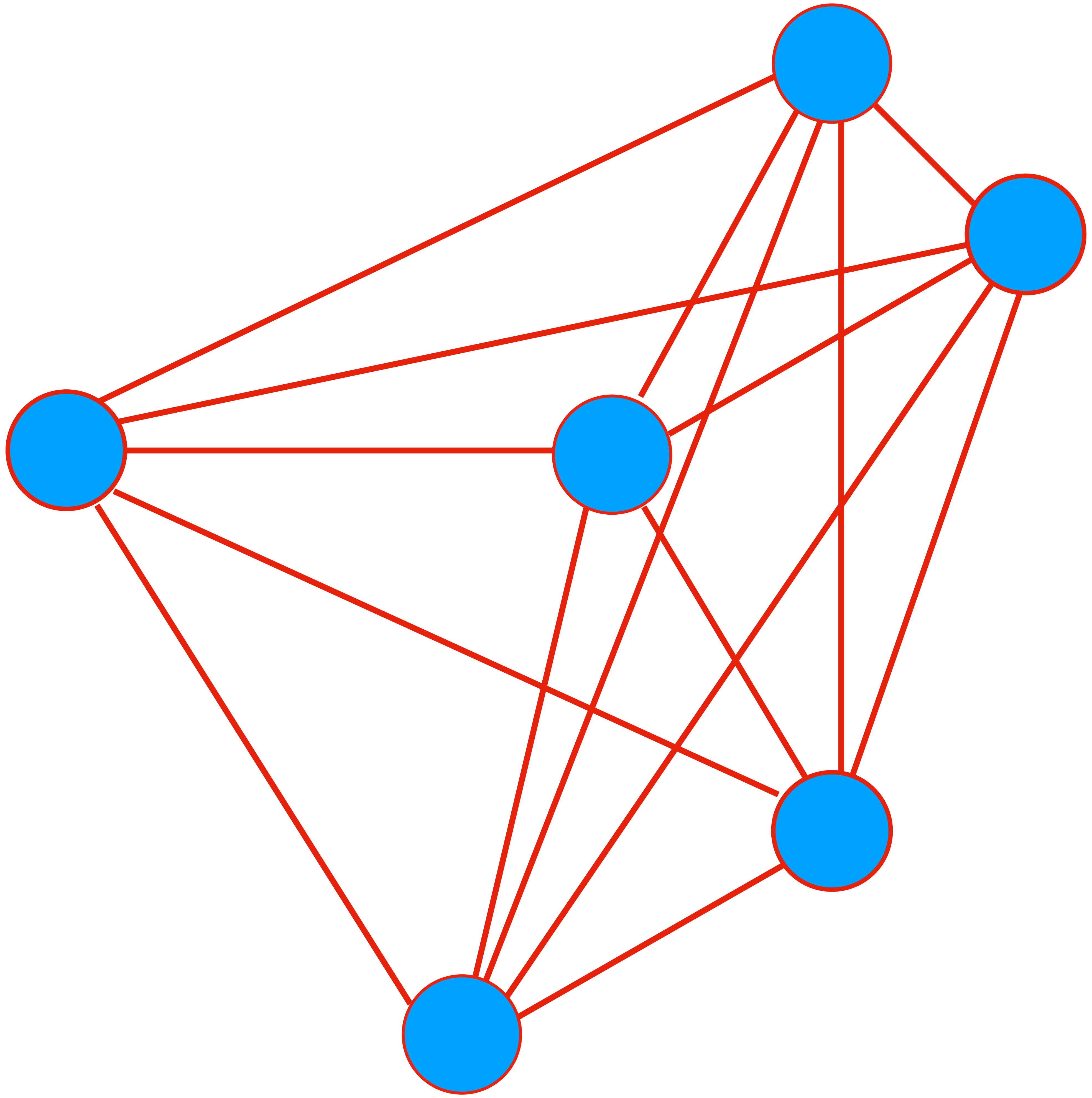


https://arxiv.org/pdf/1704.01212.pdf

European Research Council

Graph Networks

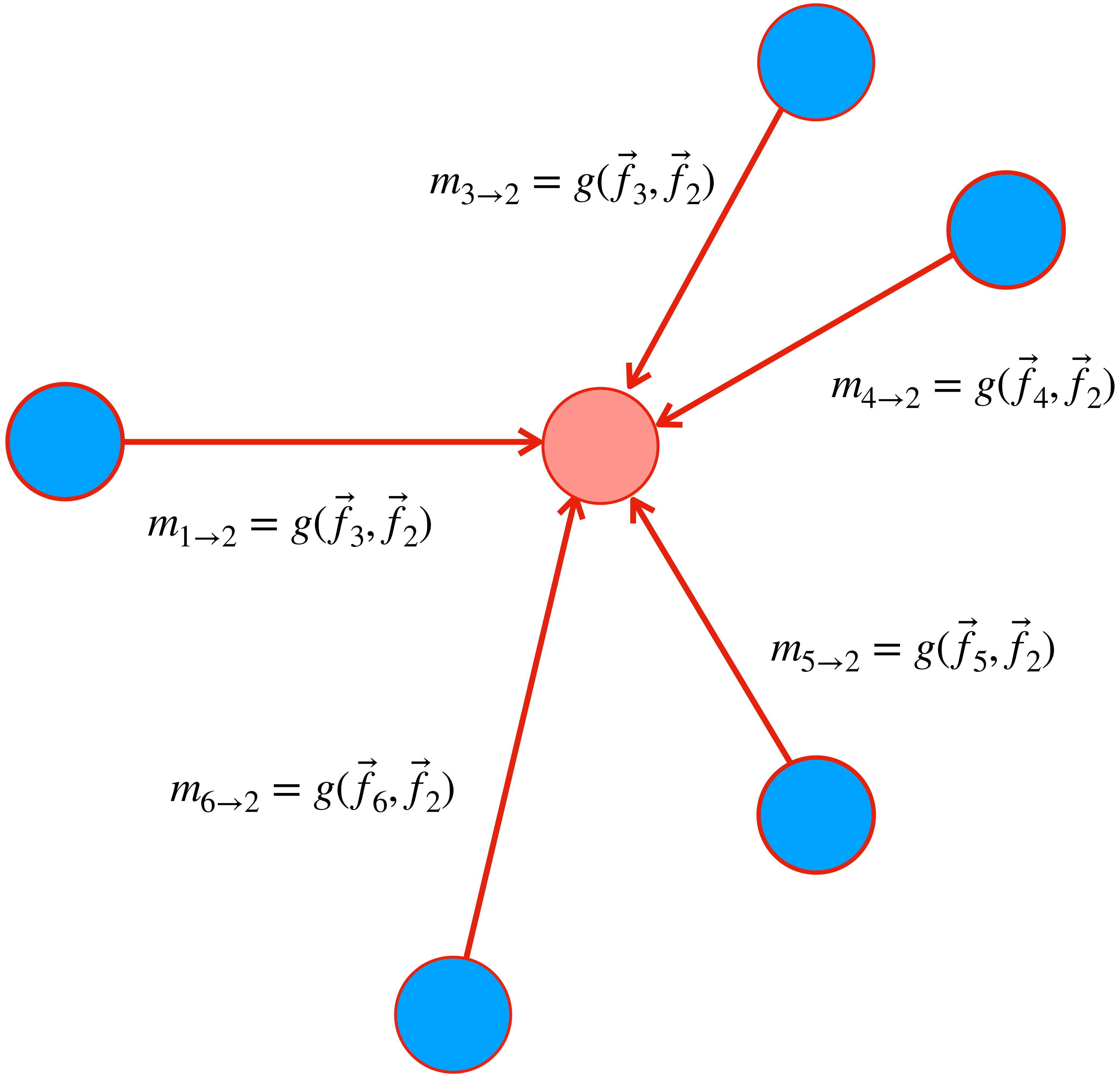
- *Once you have a graph, you want to learn from it*
  - *Each item in a dataset is represented as a set of vertices (like pixels in an image)*
  - *Each vertex is represented by a vector of features (like RGB indices for images*
- ***Vertices are connected through links***



https://arxiv.org/pdf/1704.01212.pdf

Graph Networks

- *Once you have a graph, you want to learn from it*
  - *Each item in a dataset is represented as a set of vertices (like pixels in an image)*
  - *Each vertex is represented by a vector of features (like RGB indices for images*
- *Vertices are connected through links*
- ***Messages are passed through links and aggregated on the vertices***

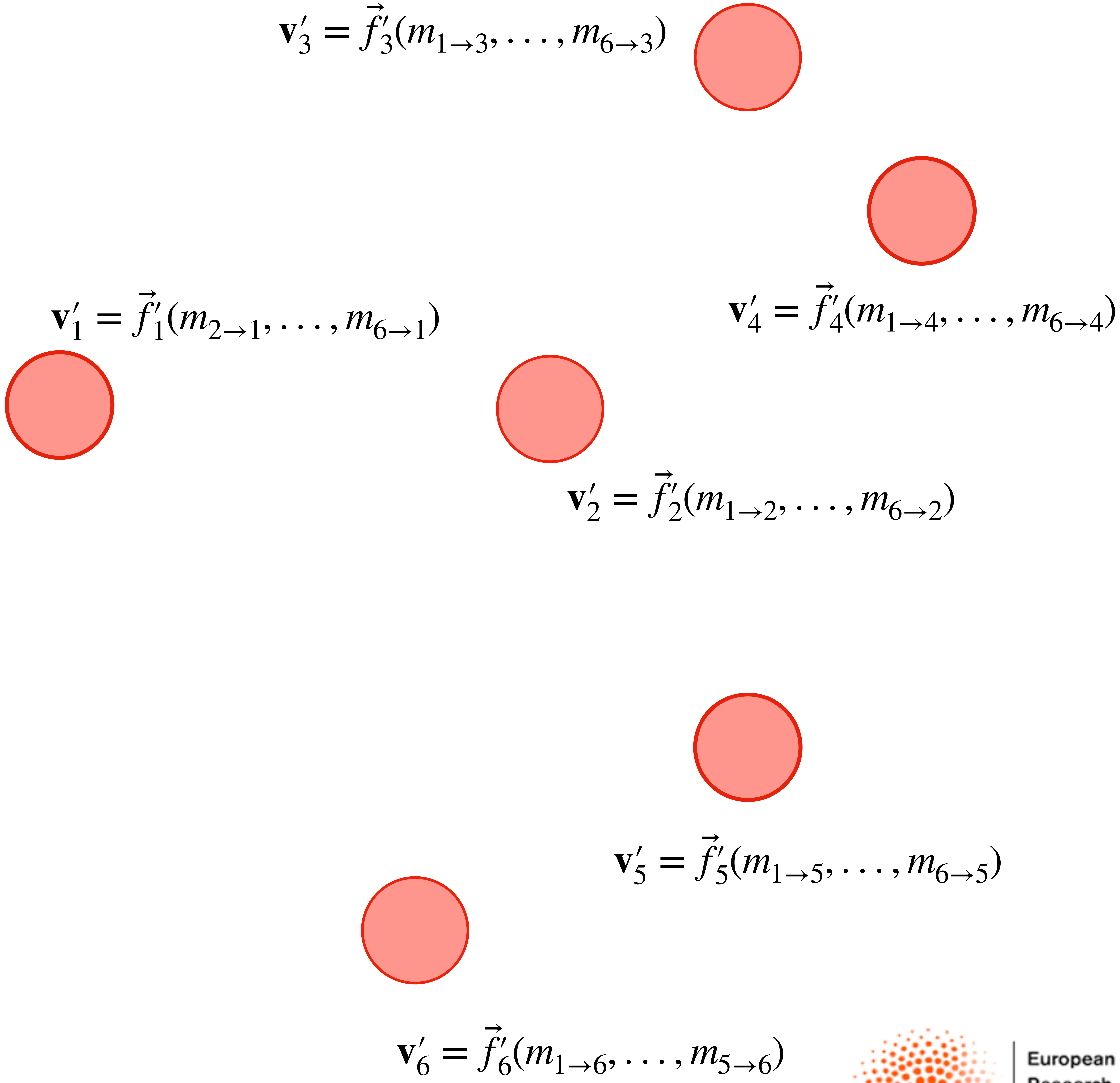


https://arxiv.org/pdf/1704.01212.pdf

European Research Council

Graph Networks

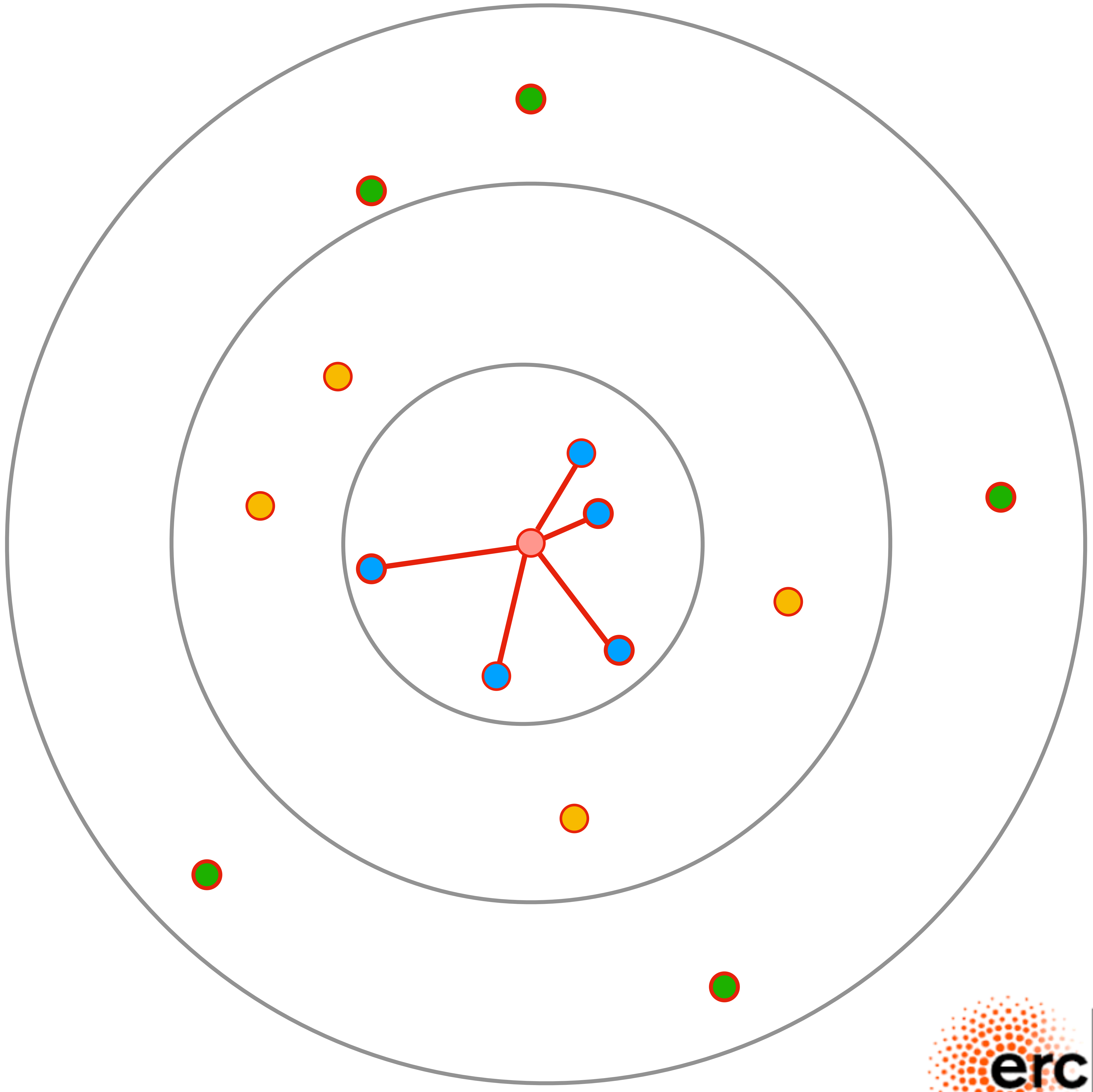
- *Once you have a graph, you want to learn from it*
  - *Each item in a dataset is represented as a set of vertices (like pixels in an image)*
  - *Each vertex is represented by a vector of features (like RGB indices for images*
- *Vertices are connected through links*
- *Messages are passed through links and aggregated on the vertices*
- ***A new representation of each node is created, based on the information gathered across the graph***



https://arxiv.org/pdf/1704.01212.pdf

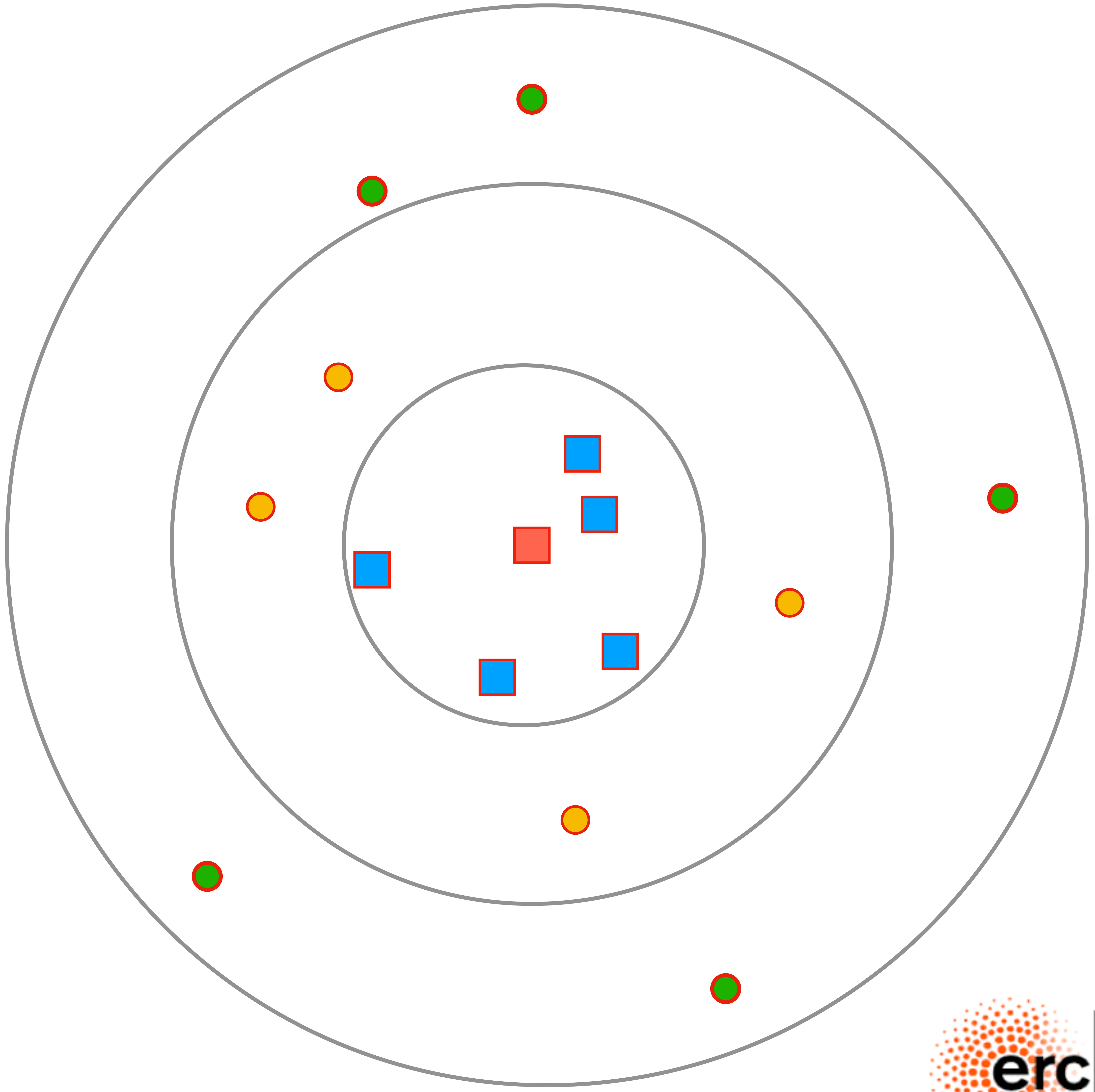
...and repeat

- *Take the case of a locally-connected graph*
  - ***At first step, only near neighbours are considered***
  - *The first message passing creates a new representation*
  - *Then you could connect to more far-away vertices*
  - *And obtain a new representation of the vertices*
  - *etc etc…*
- *This new representation emerges collectively from the graph, not just from the vertex it refers to*



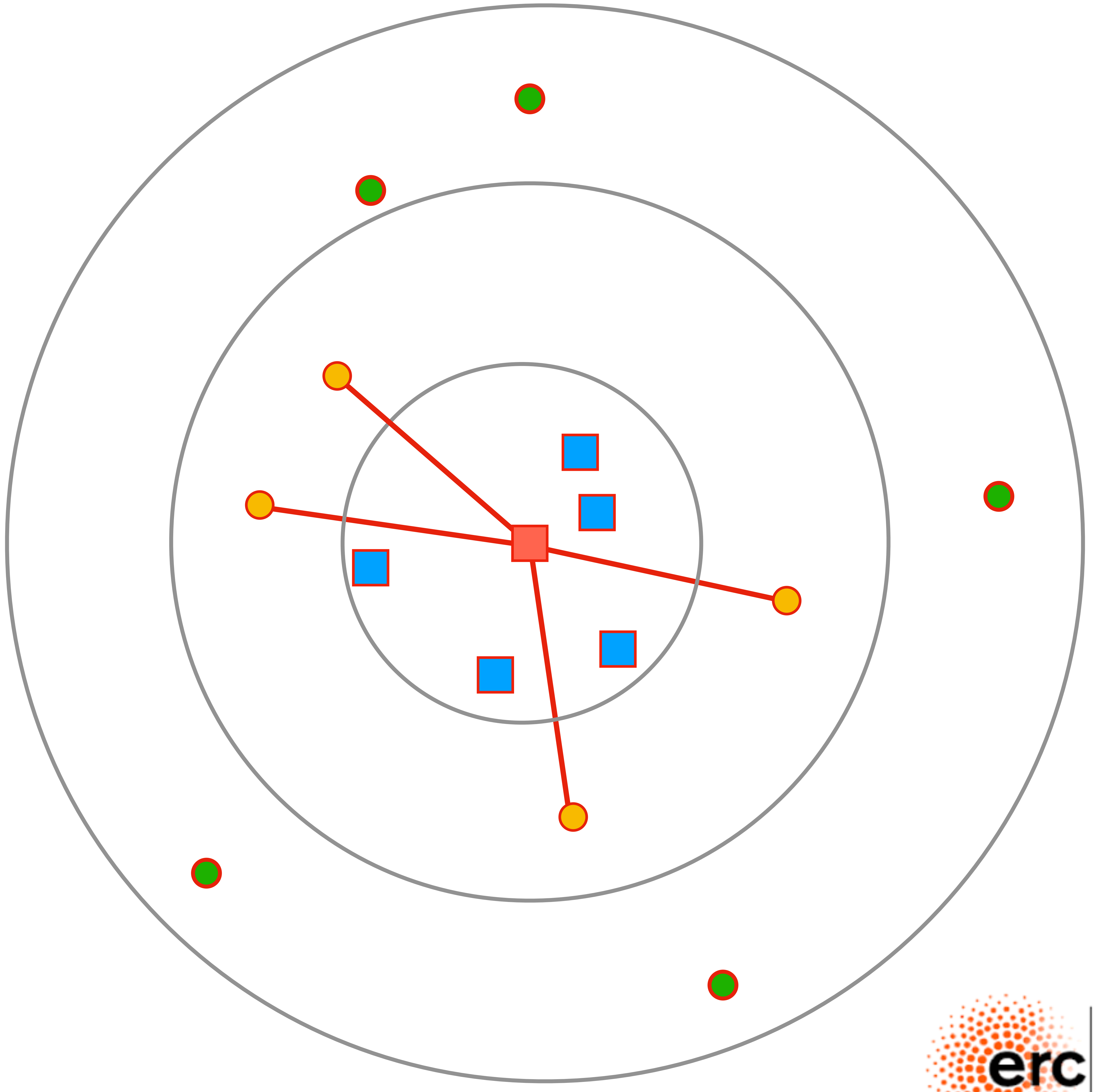
...and repeat

- Take the case of a locally-connected graph
  - At first step, only near neighbours are considered
  - **The first message passing creates a new representation**
  - Then you could connect to more far-away vertices
  - And obtain a new representation of the vertices
  - etc etc…
- This new representation emerges collectively from the graph, not just from the vertex it refers to



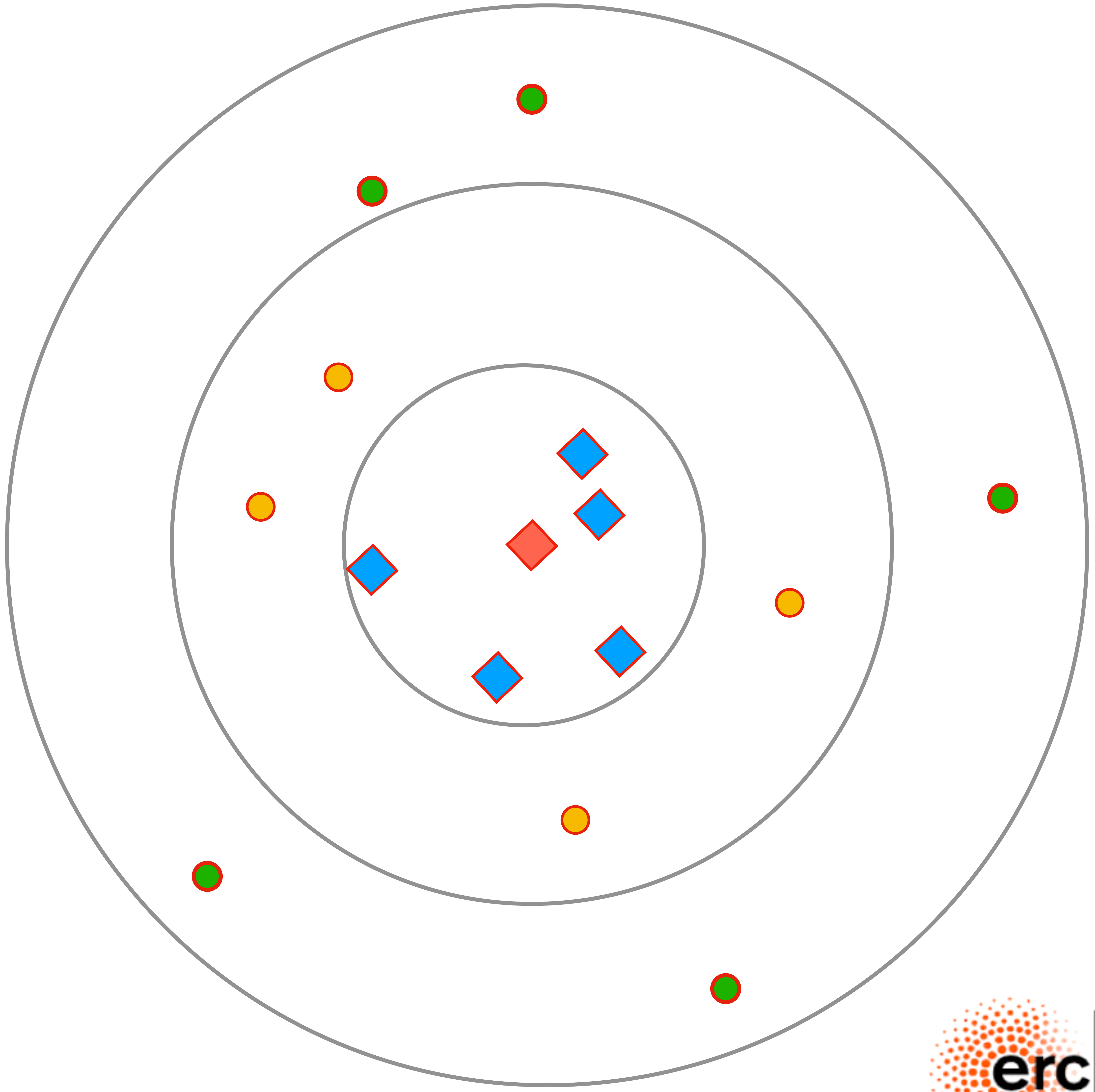
...and repeat

- *Take the case of a locally-connected graph*
  - *At first step, only near neighbours are considered*
  - *The first message passing creates a new representation*
  - ***Then you could connect to more far-away vertices***
  - *And obtain a new representation of the vertices*
  - *etc etc…*
- *This new representation emerges collectively from the graph, not just from the vertex it refers to*



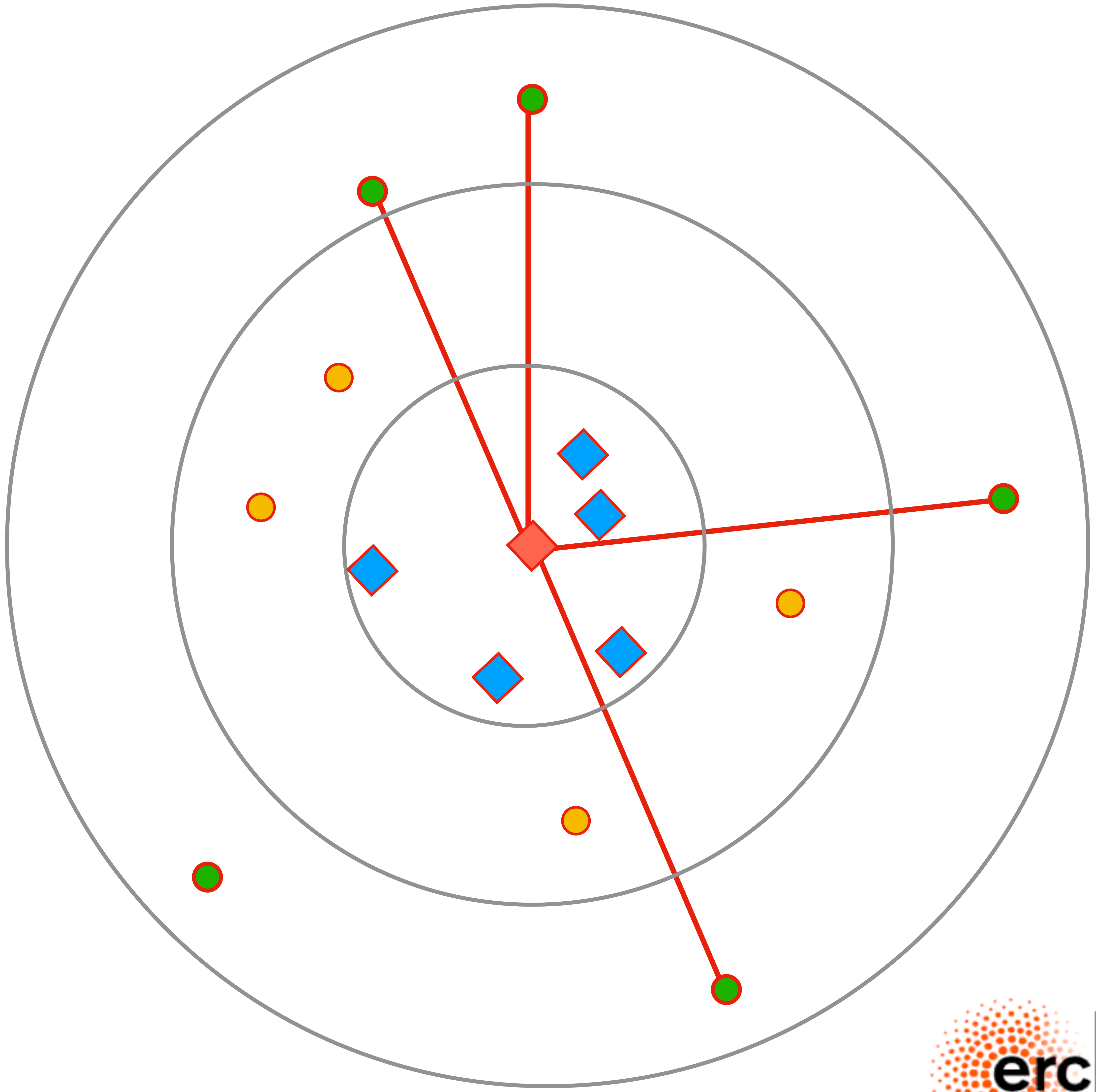
...and repeat

- *Take the case of a locally-connected graph*
  - *At first step, only near neighbours are considered*
  - *The first message passing creates a new representation*
  - *Then you could connect to more far-away vertices*
  - ***And obtain a new representation of the vertices***
  - *etc etc…*
- *This new representation emerges collectively from the graph, not just from the vertex it refers to*



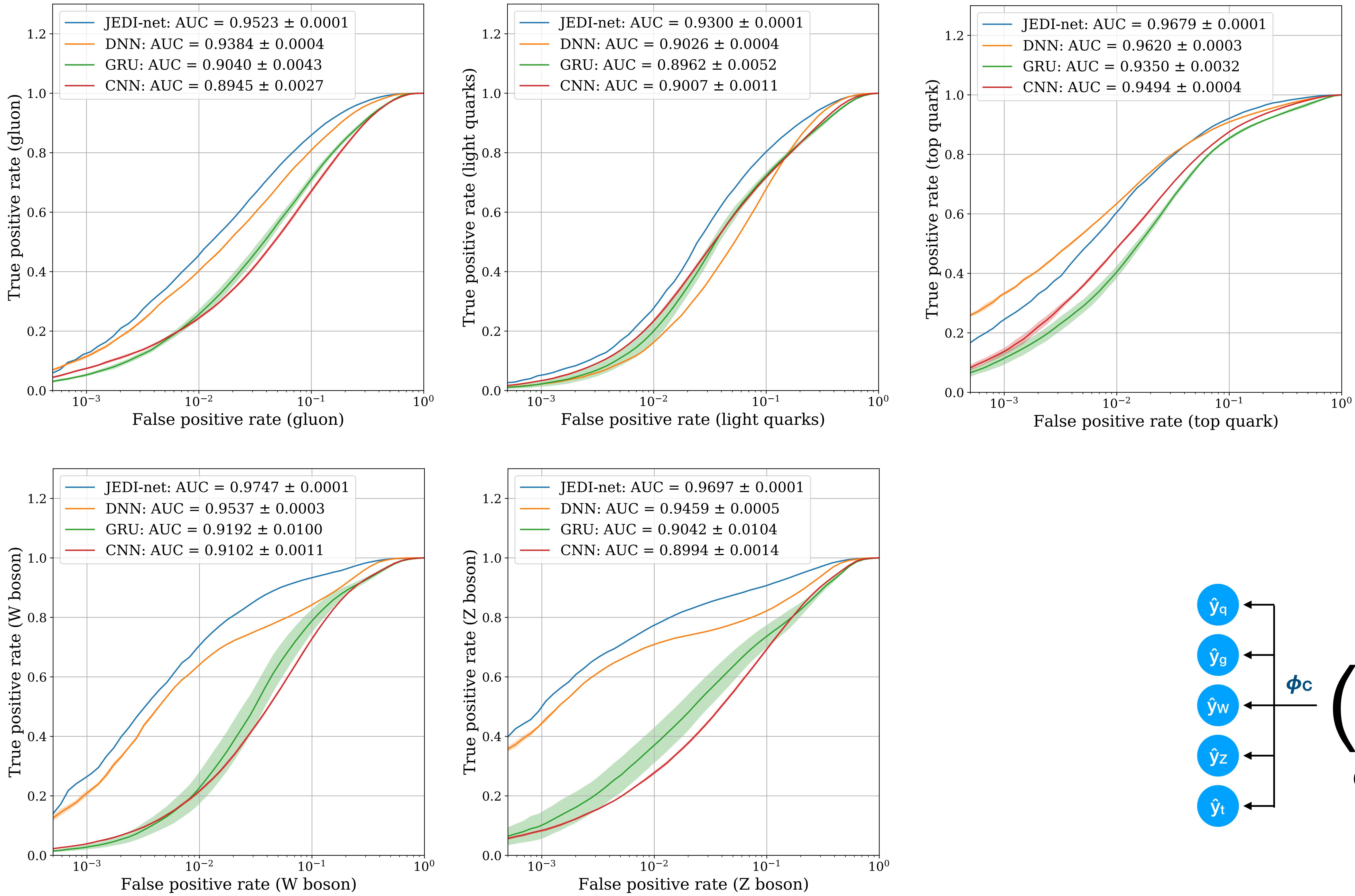
...and repeat

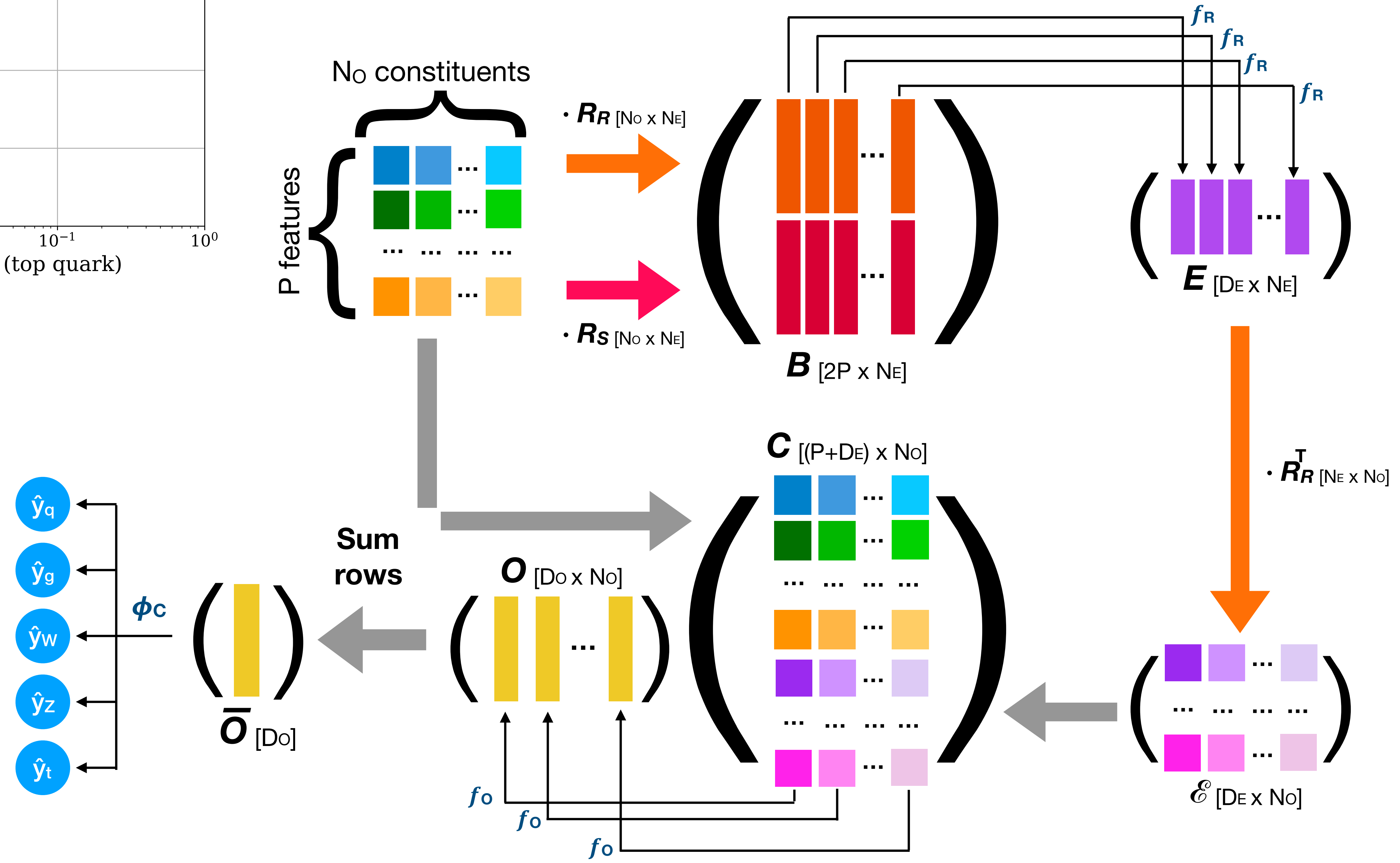
- *Take the case of a locally-connected graph*
  - *At first step, only near neighbours are considered*
  - *The first message passing creates a new representation*
  - *Then you could connect to more far-away vertices*
  - *And obtain a new representation of the vertices*
  - ***etc etc…***
- ***This new representation emerges collectively from the graph, not just from the vertex it refers to***



European Research Council

It works!





N_O: # of constituents
P: # of features
$N_E = N_O(N_O-1)$: # of edges

ϕ_C, f_O, f_R parameterize

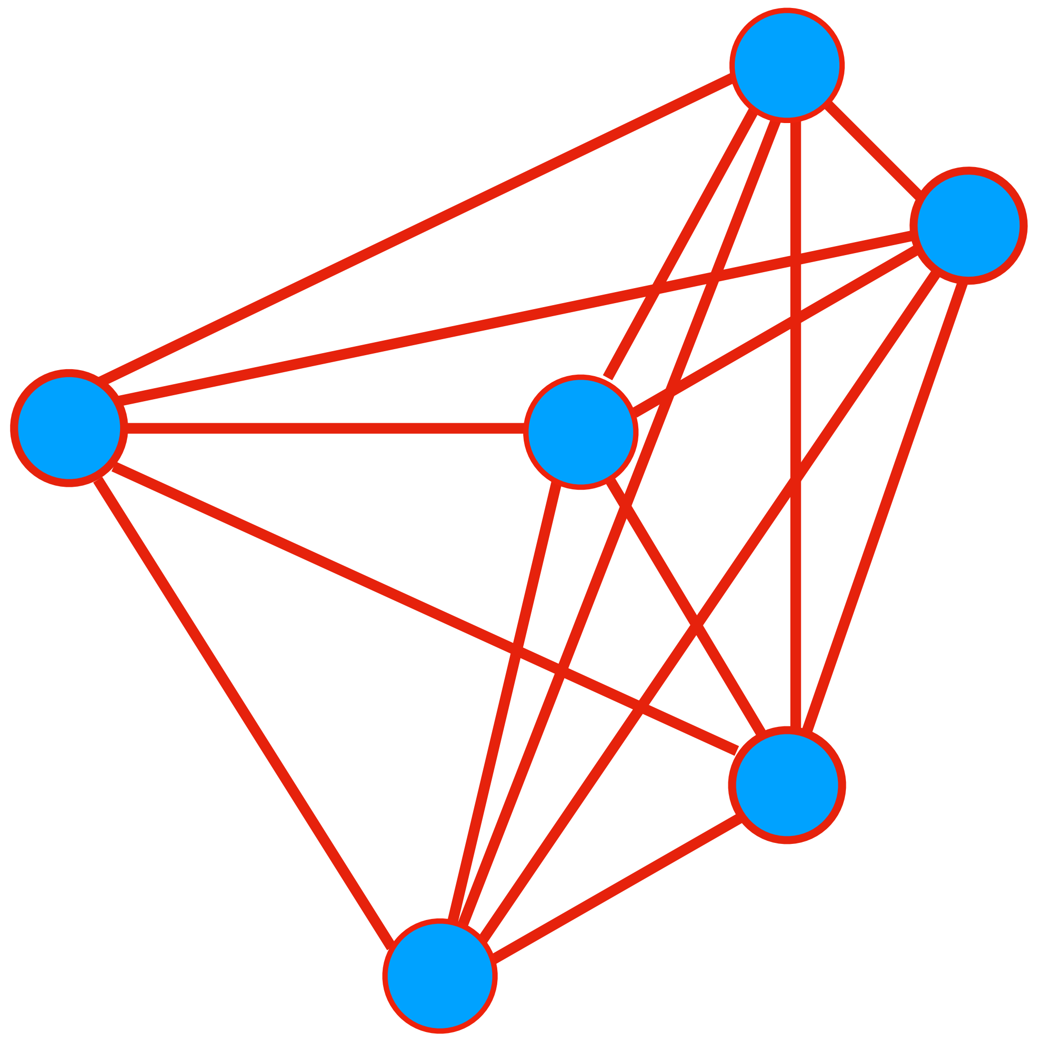
Your hands-on exercise, with Graph NNs (and more data)

European Research Council

The math

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f \ (nodes \times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c \ (feature \ weight \times channels)}$$

- *The inputs X*
- *The weights W*
- *The Adjacency matrix*



The Inputs

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n\times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n\times f \ (nodes\times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f\times c \ (feature \ weight\times channels)}$$

- *Same as all other networks*
  - *Each vertex (row) is represented as an array of features (columns)*

The Weights

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f \ (nodes \times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c \ (feature \ weight \times channels)}$$

- *The weight matrix W is used on each vertex to create new function of the inputs x (encoding)*
- *If wij=1, the input representations is used directly in the message passing*

The Adjacency Matrix

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f \ (nodes \times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c \ (feature \ weight \times channels)}$$

- *Embeds graph structure: says which vertex is connected to which.*
- *The value could be 1 (0 for no connection) or it could be a weight*
- *Could be used with attention mechanism: the fixed weights are replaced by learnable parameters. In training, the graph decides which connections are relevant*

European Research Council

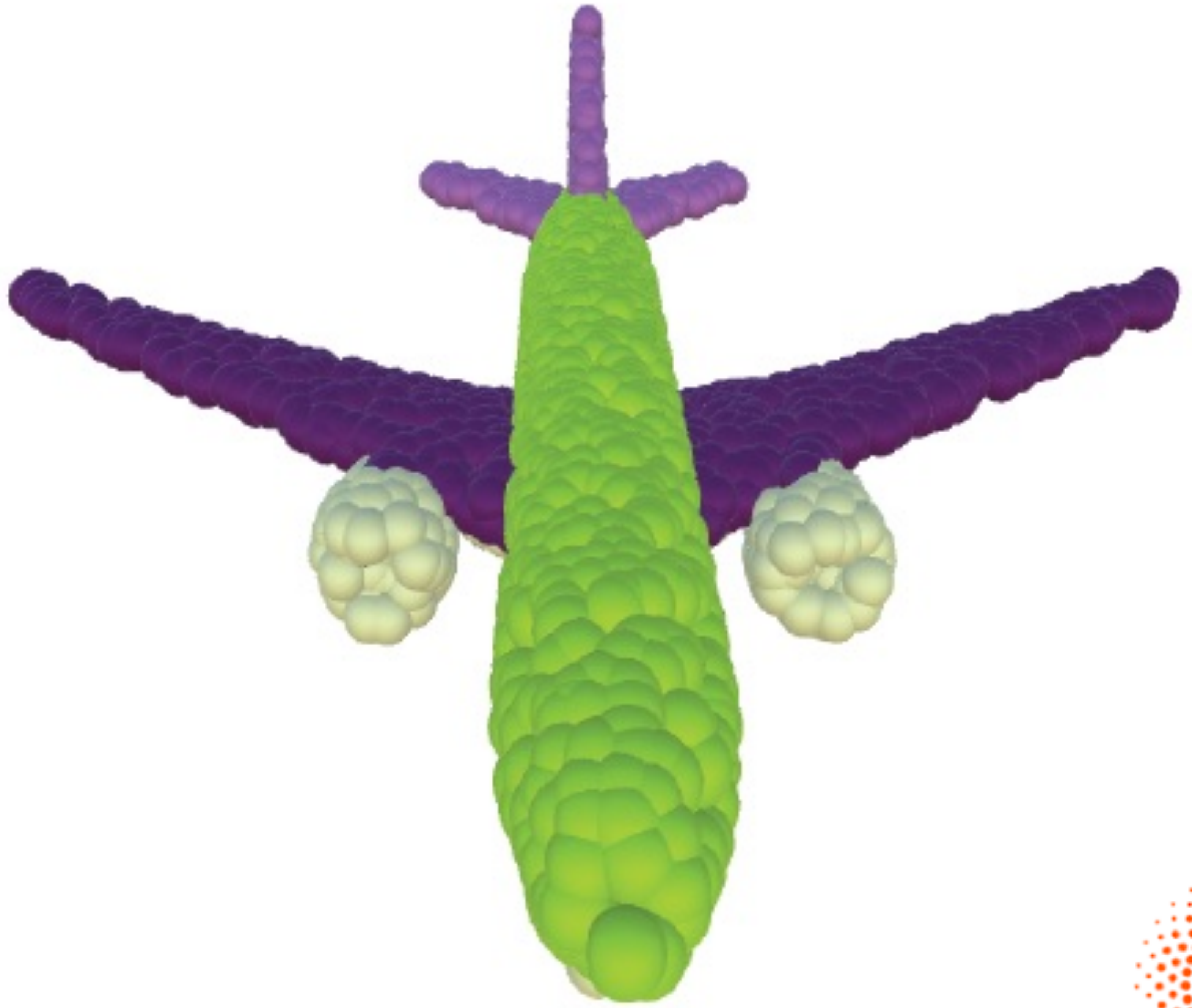
The Message Passing

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n\ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f\ (nodes \times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c\ (feature\ weight \times channels)}$$

- *By performing a standard matrix product, one builds the message*
- *This is for one filter. One can have multiple filters, as for CNNs*

EdgeConv

- *Dynamic Graph CNN (DGCNN) is one kind of message-passing neural network*
- *It uses EdgeConv layers to perform point-cloud segmentation*
- *Segmentation is the process of clustering pixels in an image into objects*
- *EdgeConv was capable of extending semantic segmentation beyond nearby-pixel clustering*
    - *the two wings of the airplane are associated to the same cluster, since they are found to be similar*

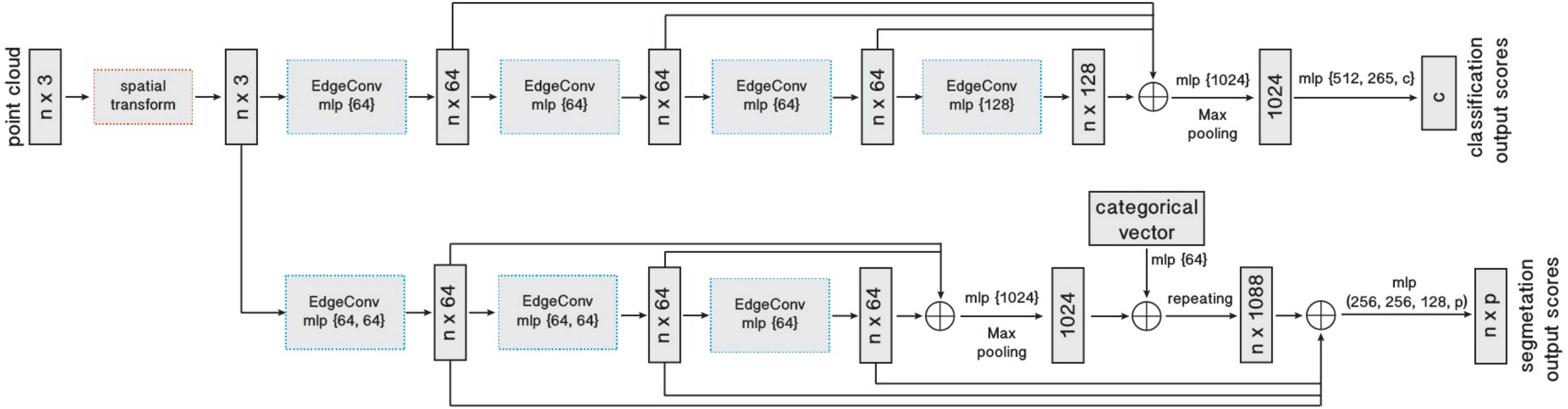


https://arxiv.org/abs/1801.07829

erc European Research Council

EdgeConv

- The actual model is much more complicated than that



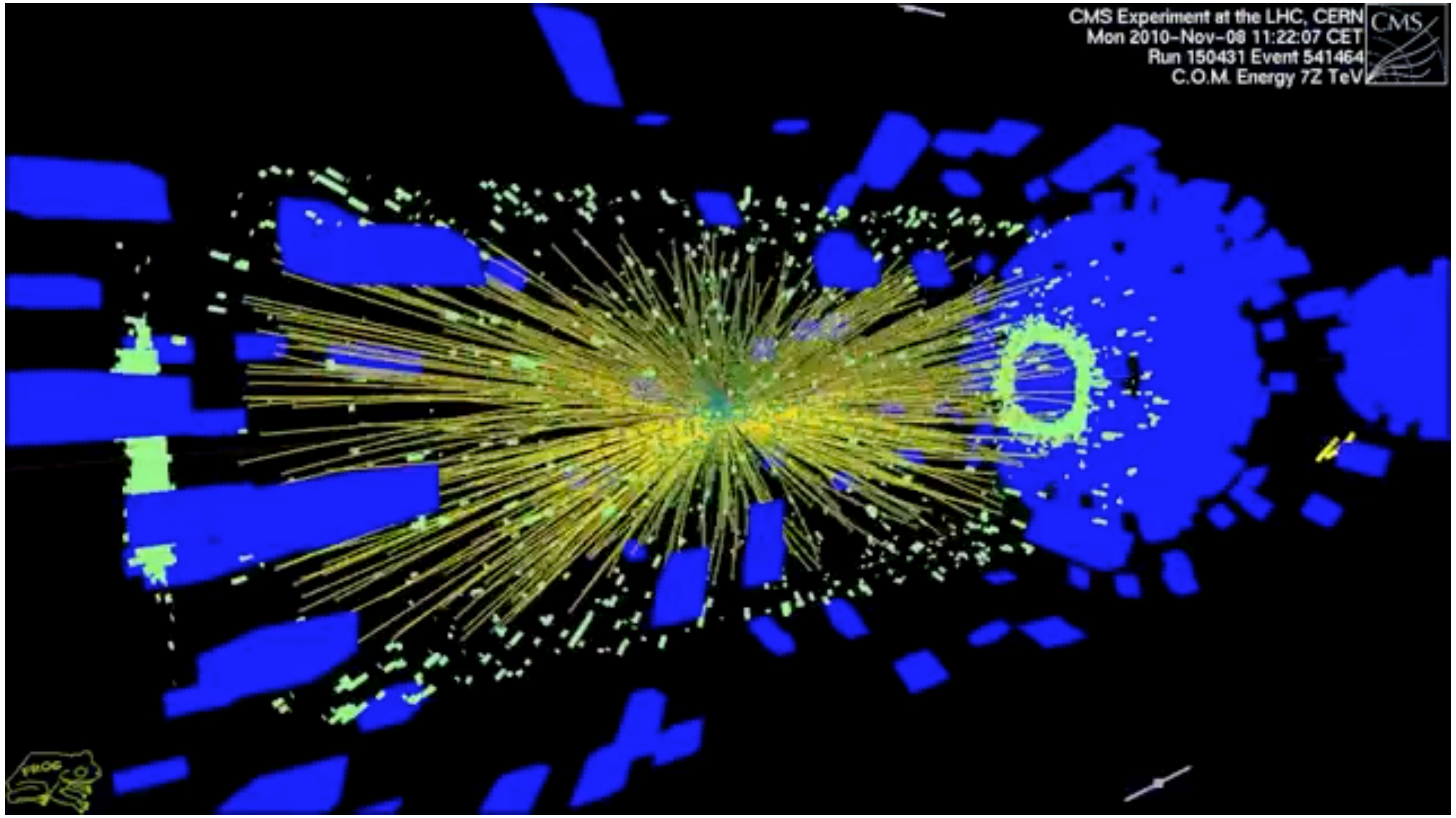
- Each EdgeConv layer runs a message passing and creates an updated representation of the graph of points

- Similar to a CNN, but capable of processing unordered sets of points

https://arxiv.org/abs/1801.07829

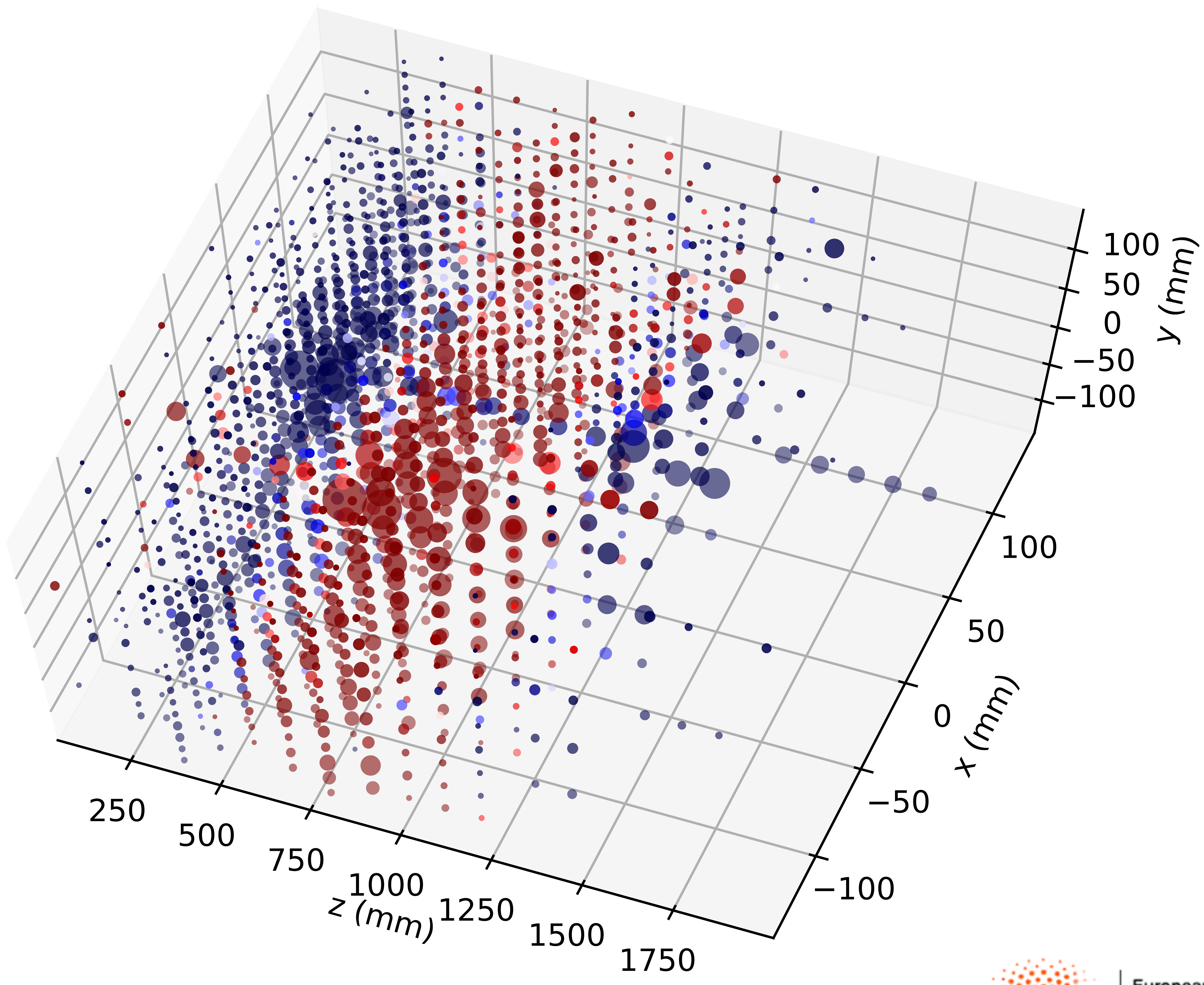
EdgeConv for Particle Physics

- DGCNN fits very well particle reconstruction in High Energy Physics
  - Particles seen as energy showers in calorimeters
  - DGCNN can be trained to distinguish overlapping showers from different particles
- Success comes at some computational cost:
  - 15 sec/event on a CPU
  - Lowered to 5 sec/event on GPU when using a batch of 100

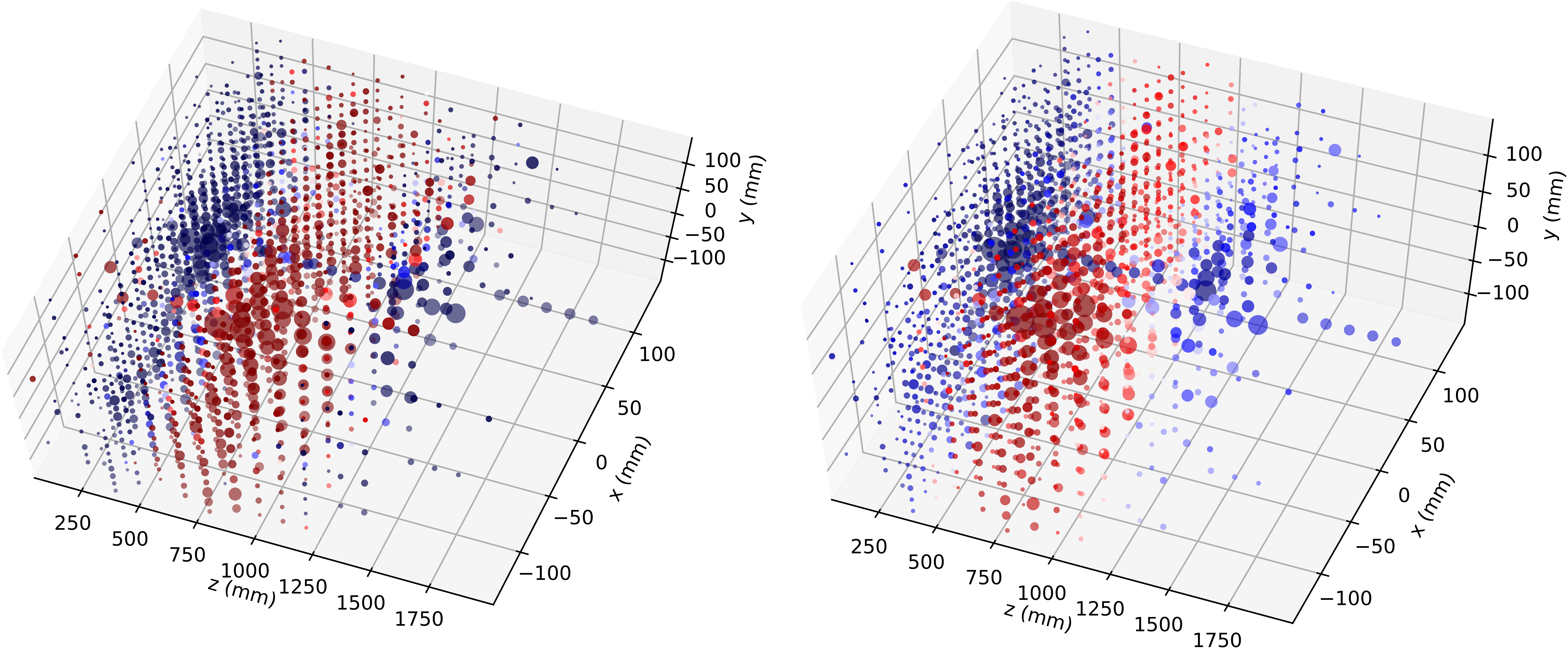


EdgeConv for Particle Physics

- *DGCNN fits very well particle reconstruction in High Energy Physics*
  - *Particles seen as energy showers in calorimeters*
  - *DGCNN can be trained to distinguish overlapping showers from different particles*
- *Success comes at some computational cost:*
  - *15 sec/event on a CPU*
  - *Lowered to 5 sec/event on GI when using a batch of 100*



Separating overlapping showers

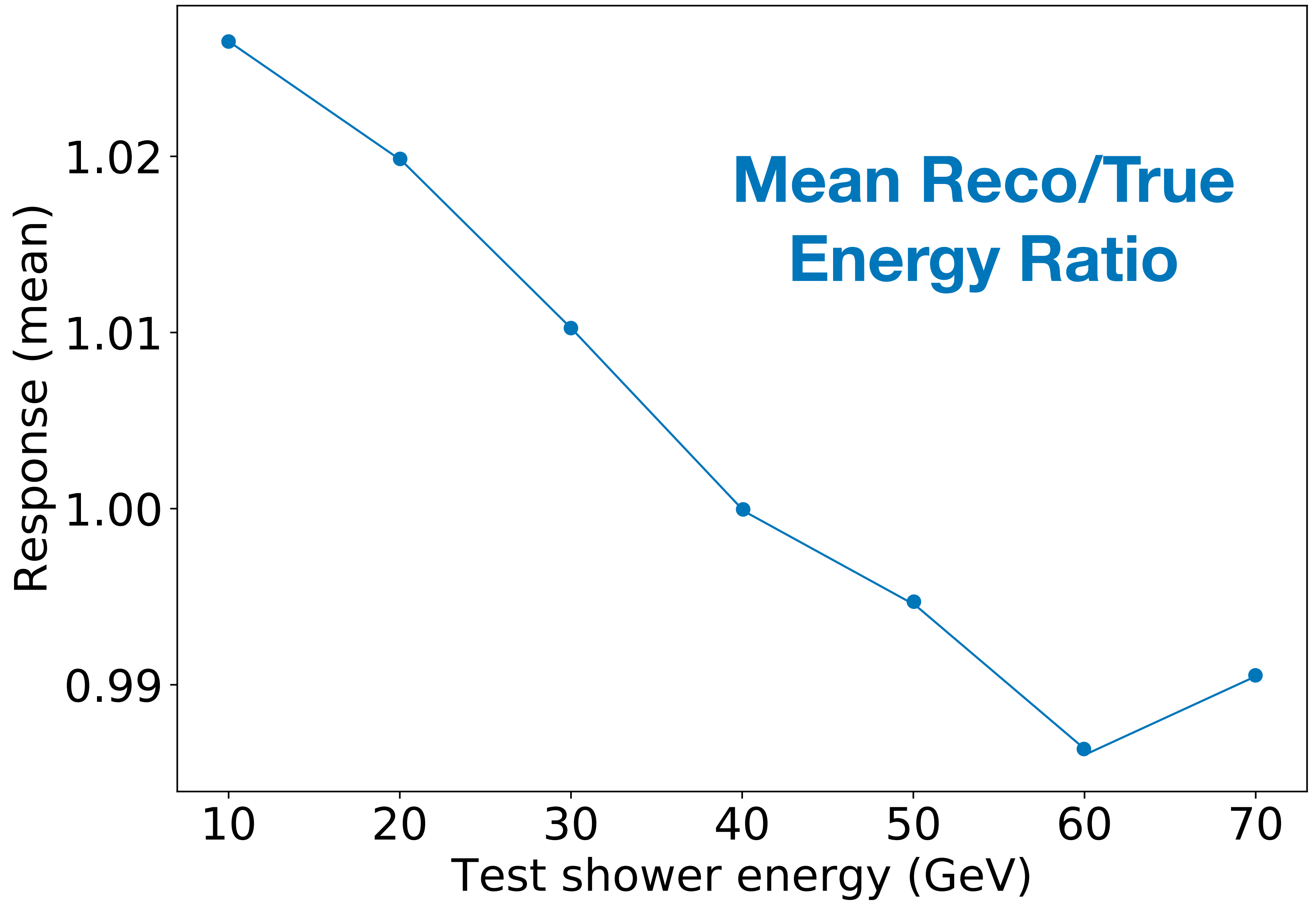


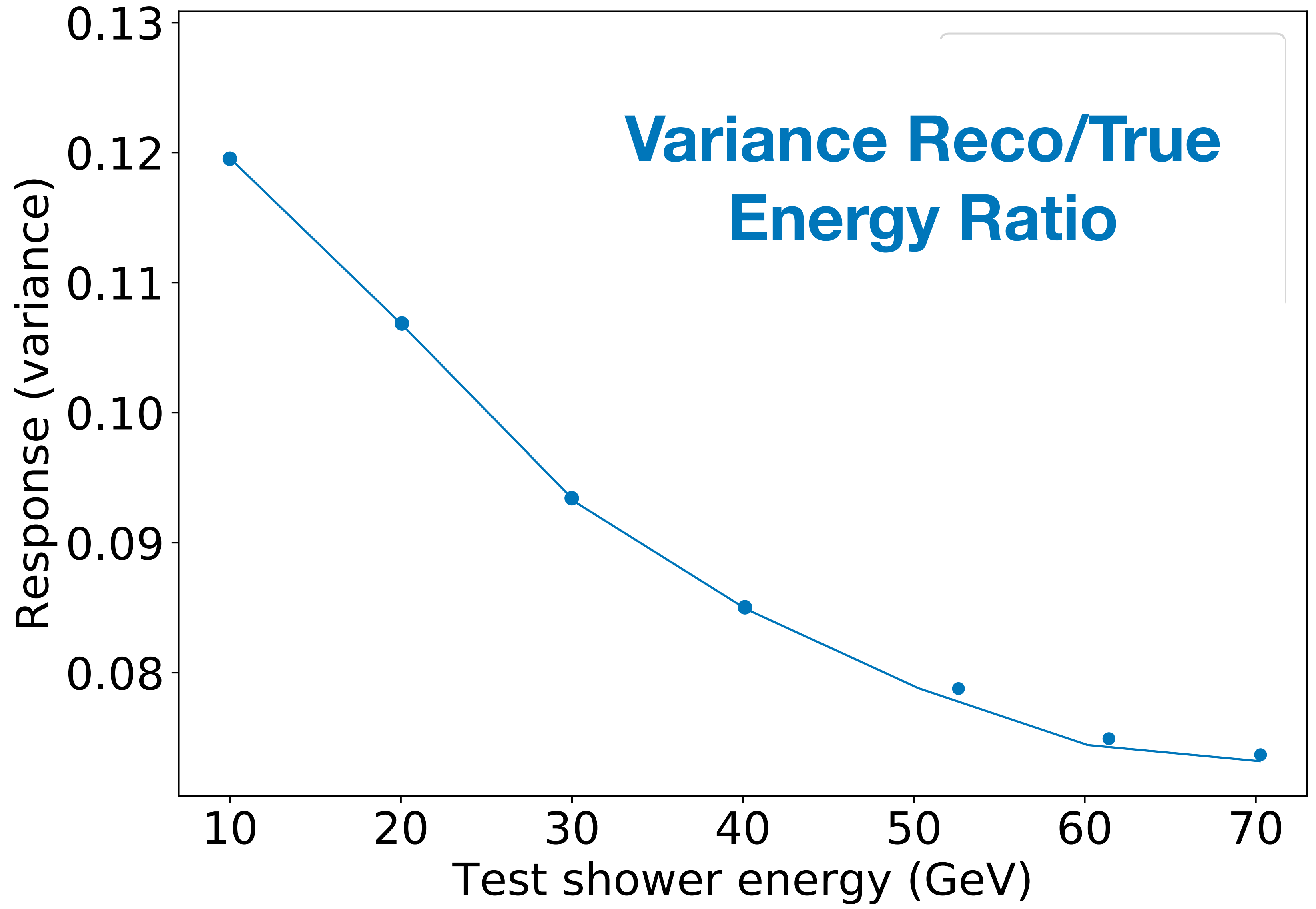
(a) Truth (b) Reconstructed

European Research Council

EdgeConv for Particle Physics

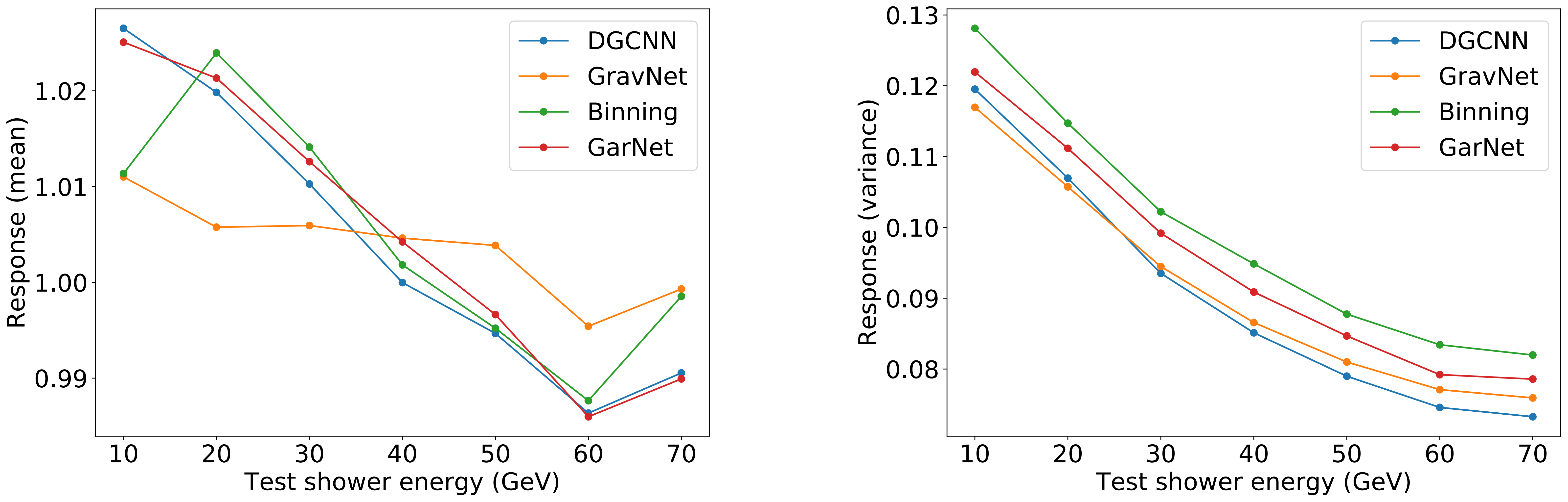
- *DGCNN fits very well particle reconstruction in High Energy Physics*
  - *Particles seen as energy showers in calorimeters*
  - *DGCNN can be trained to distinguish overlapping showers from different particles*
- *Success comes at some computational cost:*
  - *15 sec/event on a CPU*
  - *Lowered to 5 sec/event on GPU when using a batch of 100*



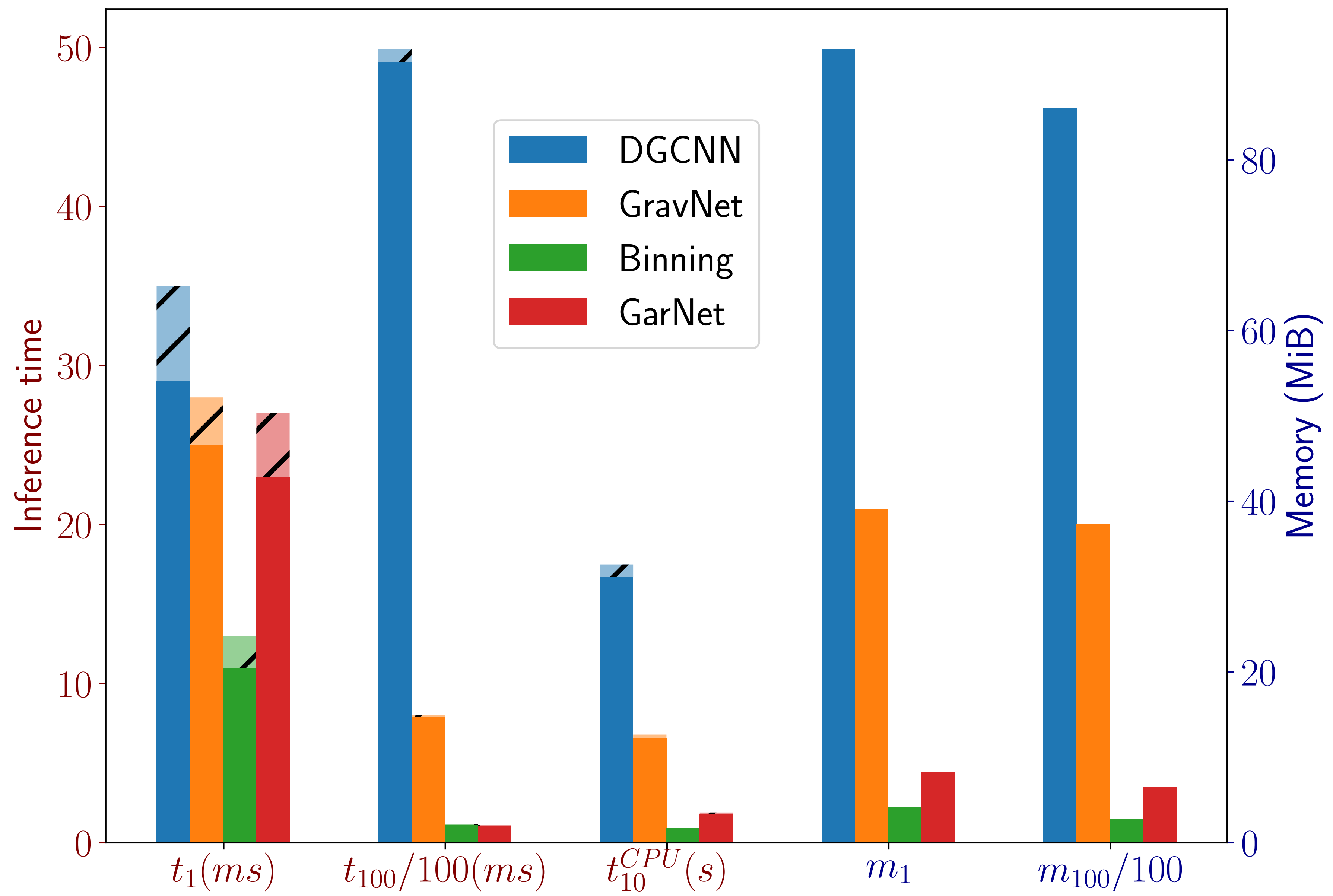


GraphNets for Calorimetry

- *Good performance achieved, comparable to more traditional approaches*
- *Using a potential (V(d)) to weight up the near neighbours allows to keep memory footprint under control (with respect to other graph approaches)*



(c) Mean (d) Variance

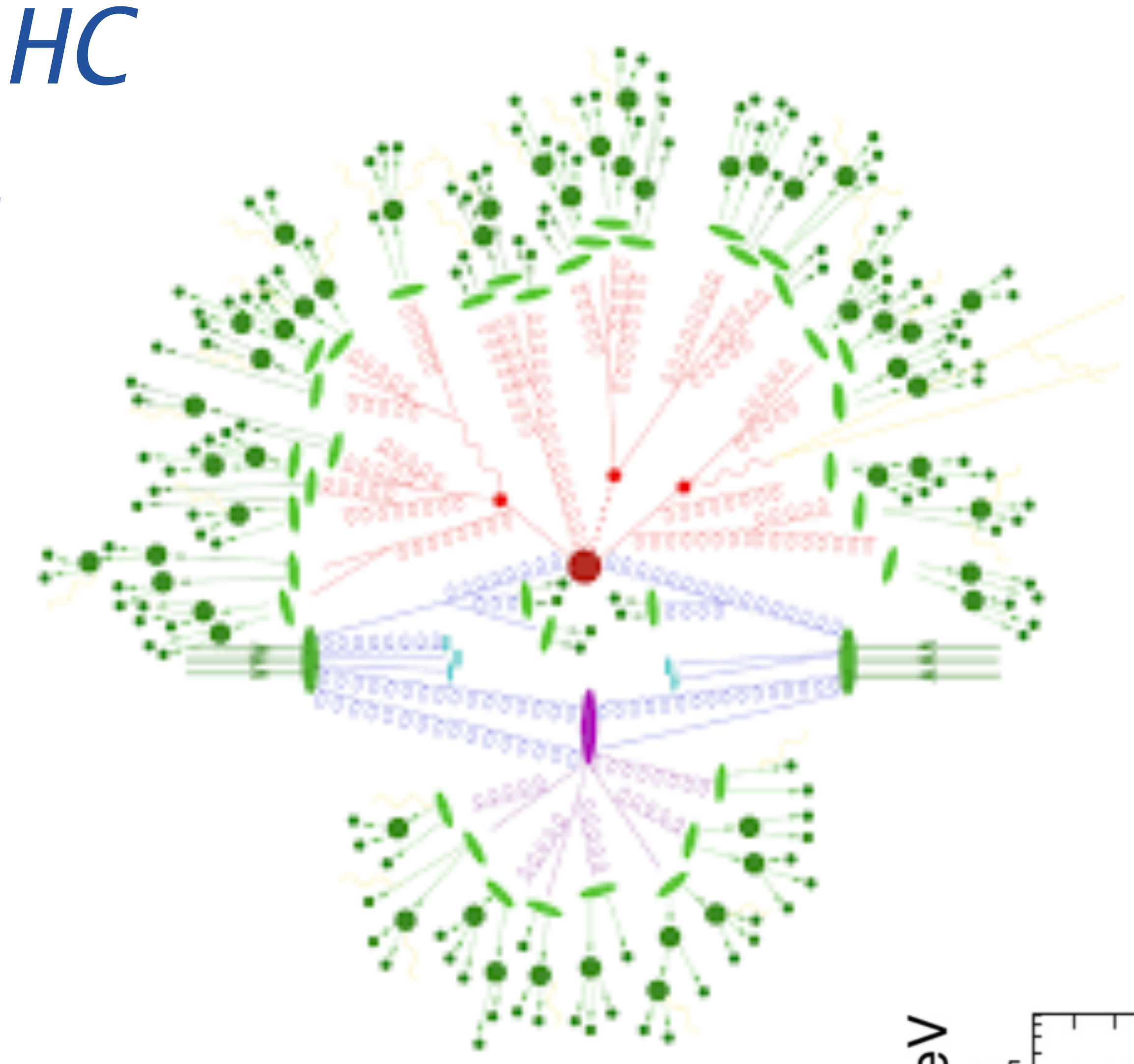


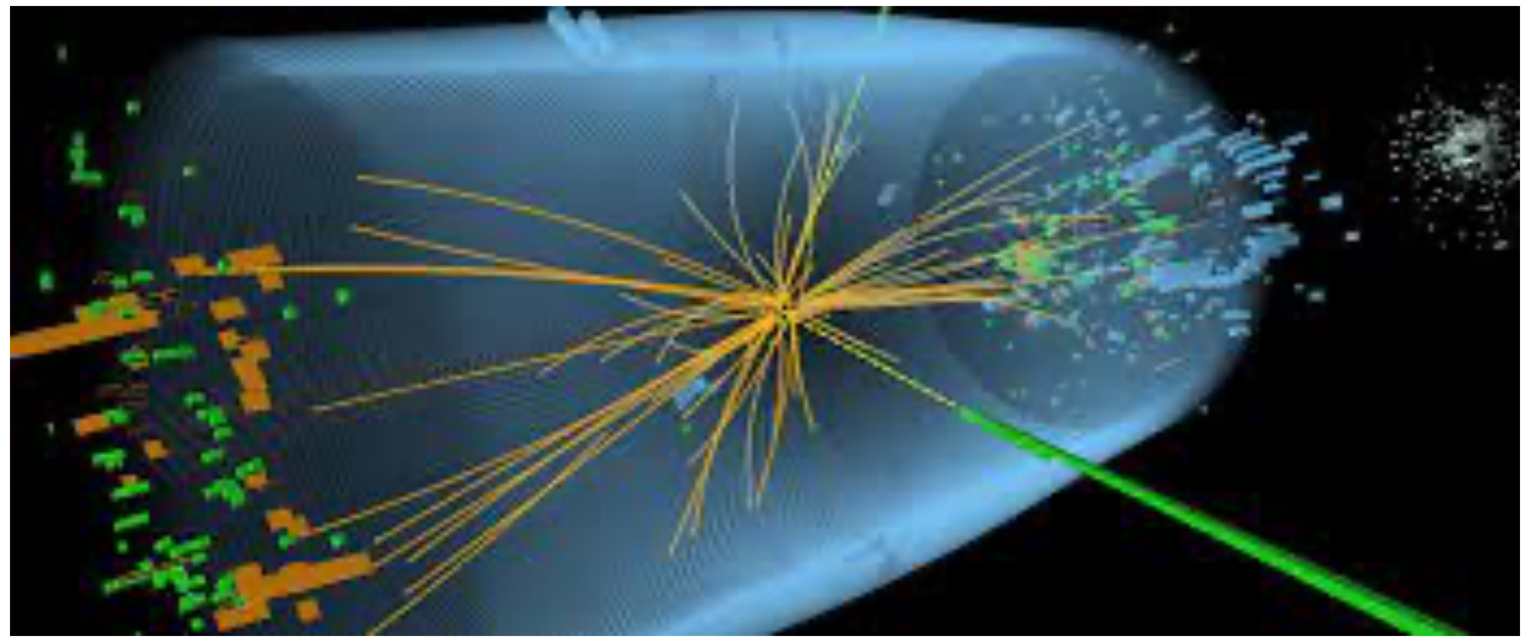
Collision Simulation with generative models

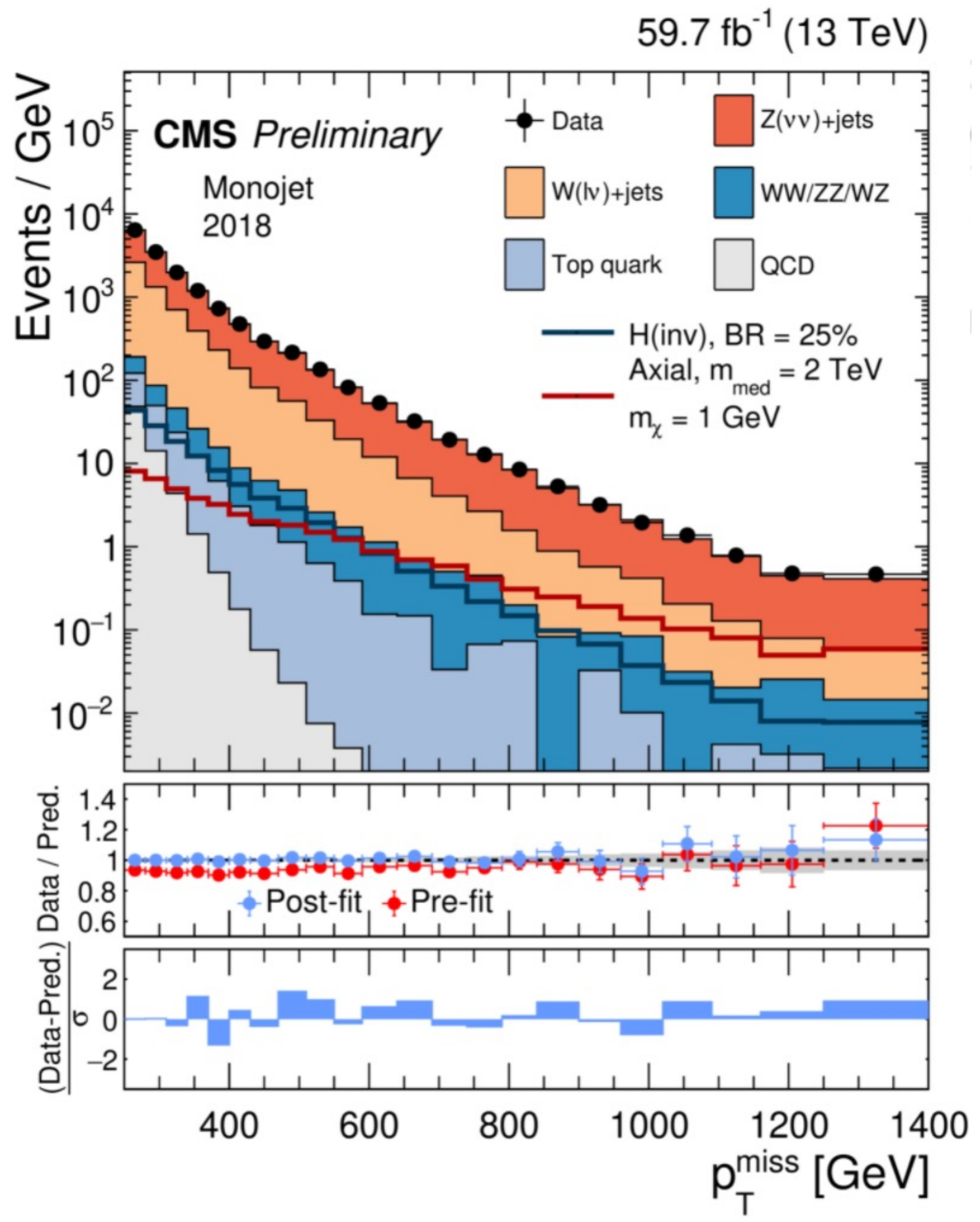
European Research Council

Why we use simulation

- The capability of simulating LHC collisions is crucial for data analysis
  - So that we can study what a given new phenomenon (e.g., dark matter produced in the collision) would look like
  - So that we can have an idea of the background we have to fight from known physics phenomena
- This is done with a set of rule-based algorithms
  - Very accurate, but very computing demanding

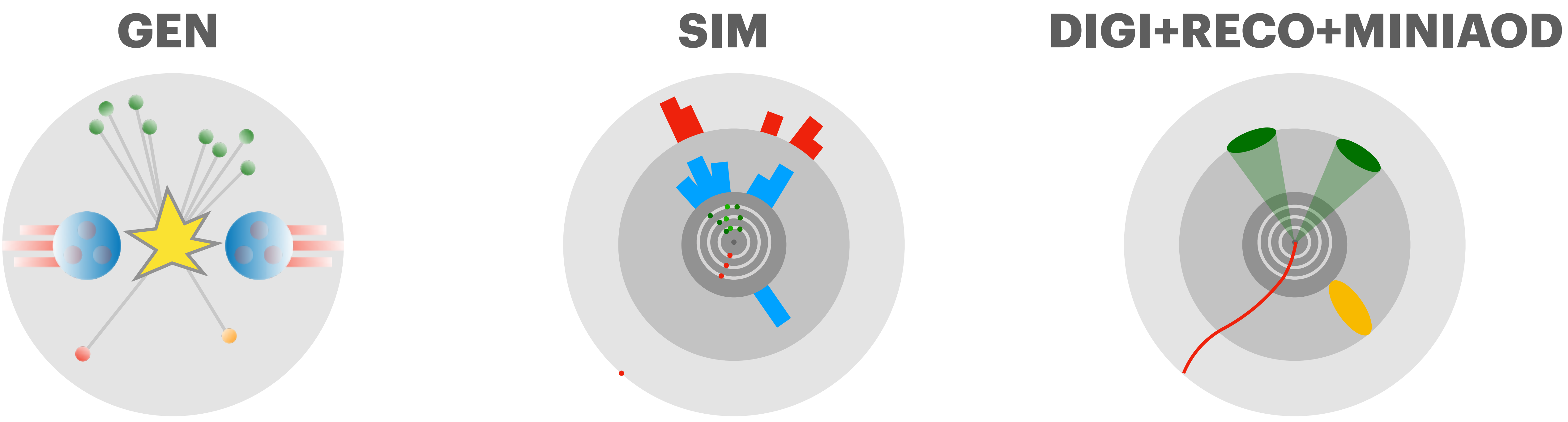


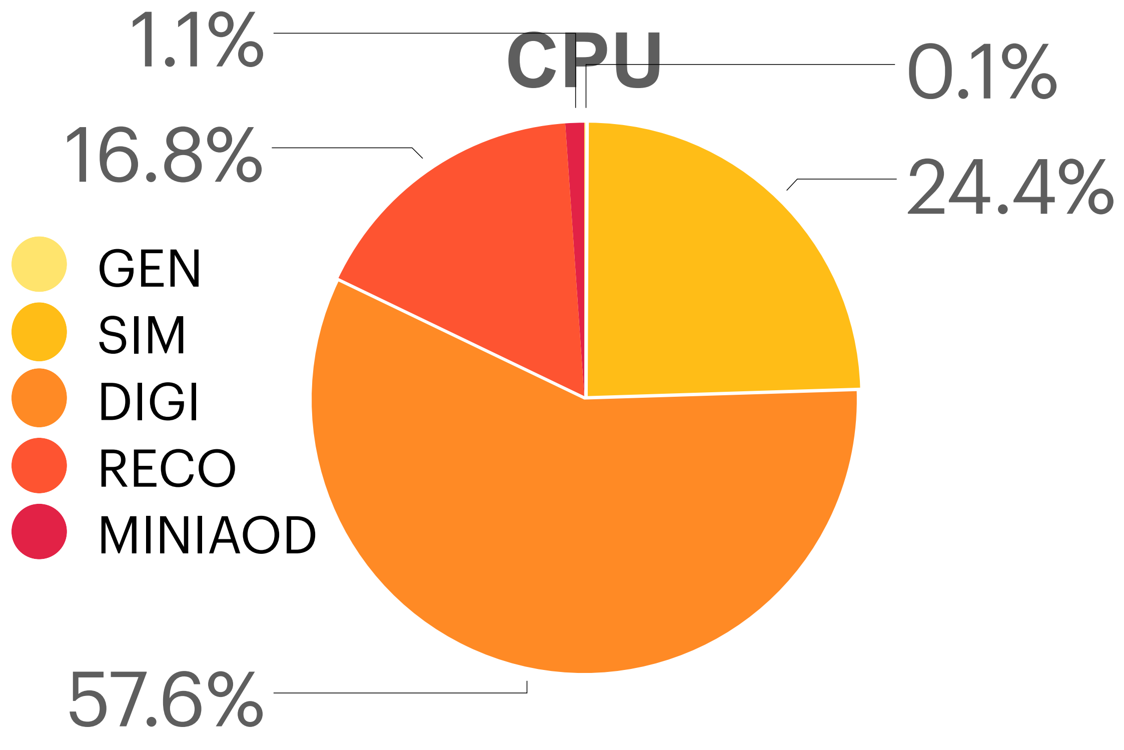


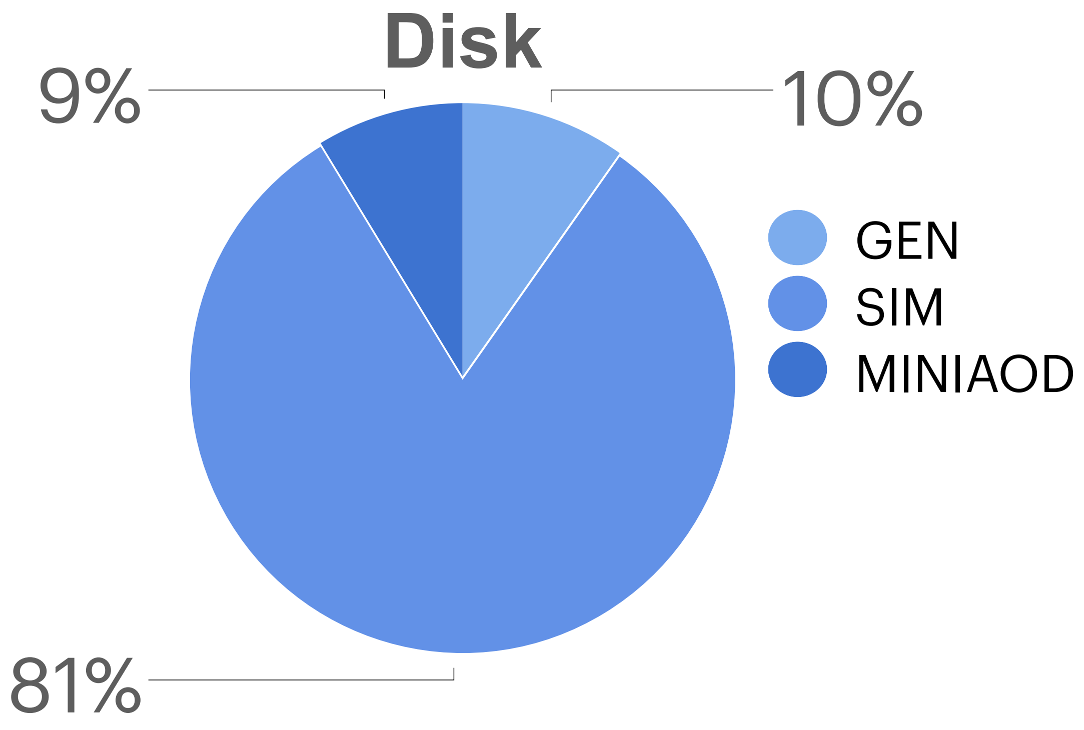


Why this is a problem

- *Large part of computing resources goes into simulating the detector response (SIM)*
- *In addition, once simulated, these data are processed as if they were real data (more CPU and Disk)*
- *Generating simulations for the whole experiment takes ~ 1 year*
  - *A tot of CPU "burned"*
  - *Disk occupied for a lot of time*
- *Because of this, we ended up taking less data than what we could (because we would not know how to process the extra data)*

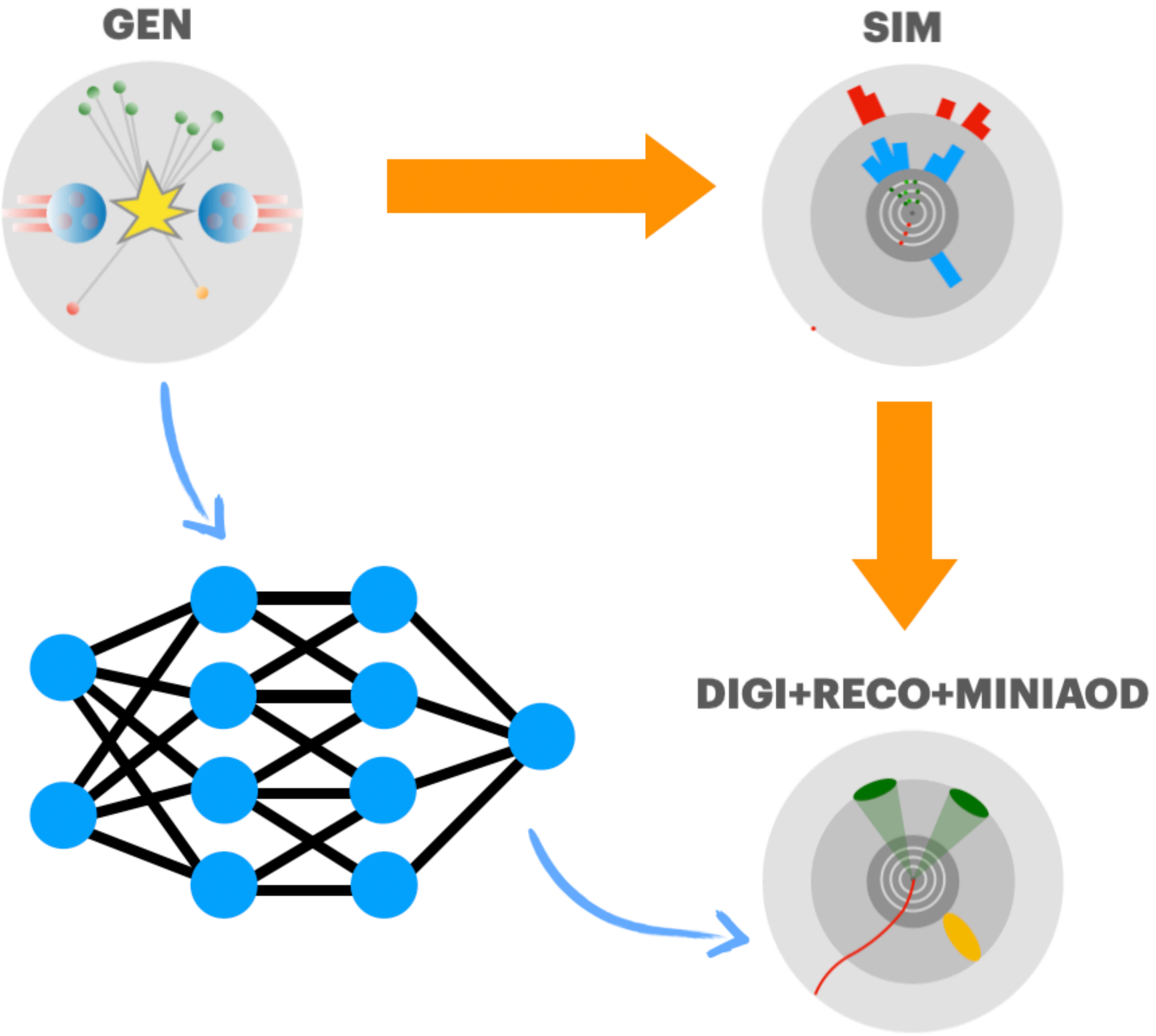






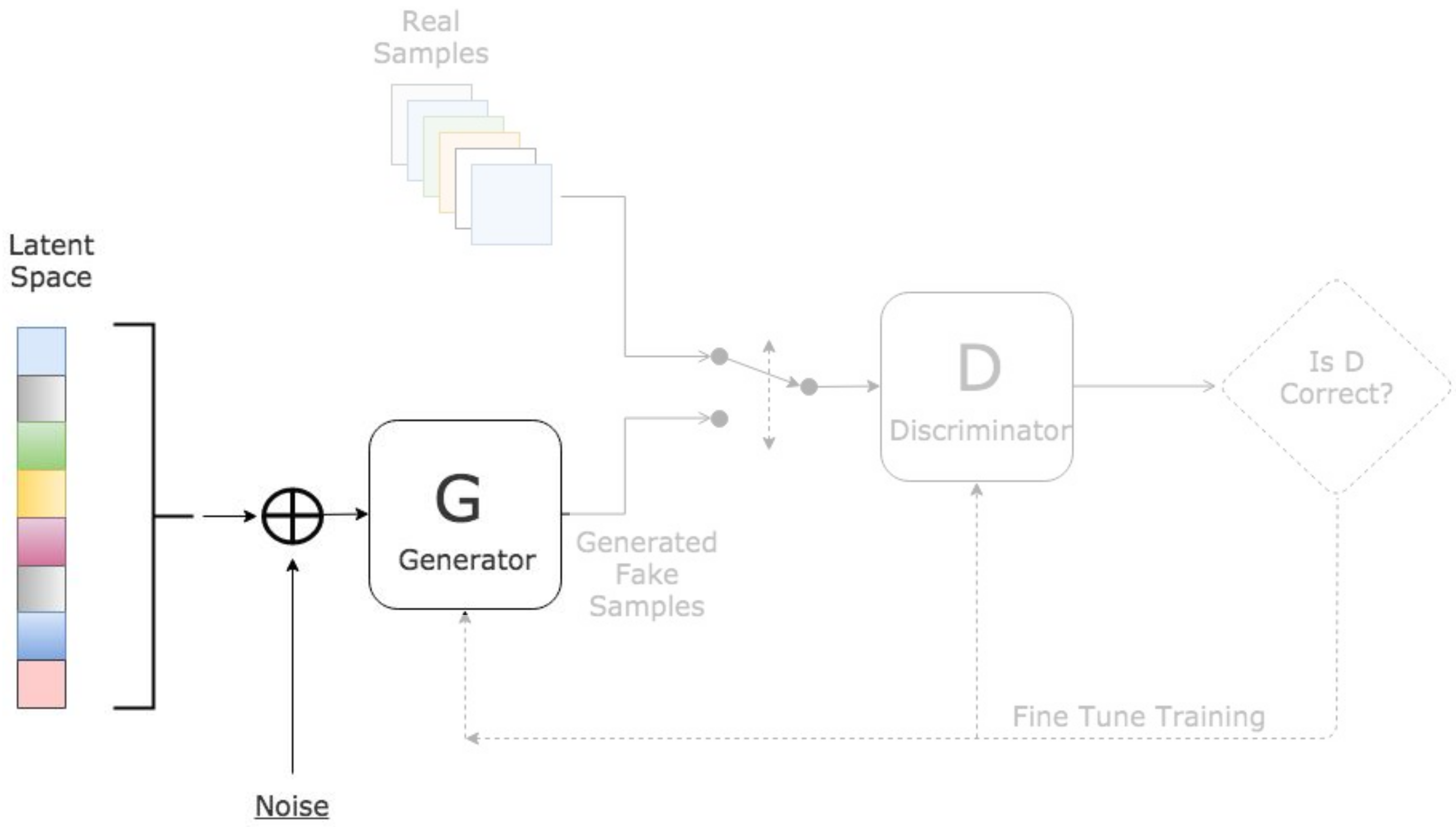
Speeding up Generation with DL

- *We have a working algorithm, accurate but slow (tens of seconds/ collision)*
- *A neural network could run in O(100 µsec)*
  - *Potential gain of a few orders of magnitude*
- *We can use data from slow algorithm to train a network to do better*



Generative Adversarial Training

- *Two networks trained against each other*
  - ***A generator aims at creating realistic data (e.g., images similar to those in the training dataset)***
  - *A discriminator aims at identifying which data in a dataset are real and which come from the generator*

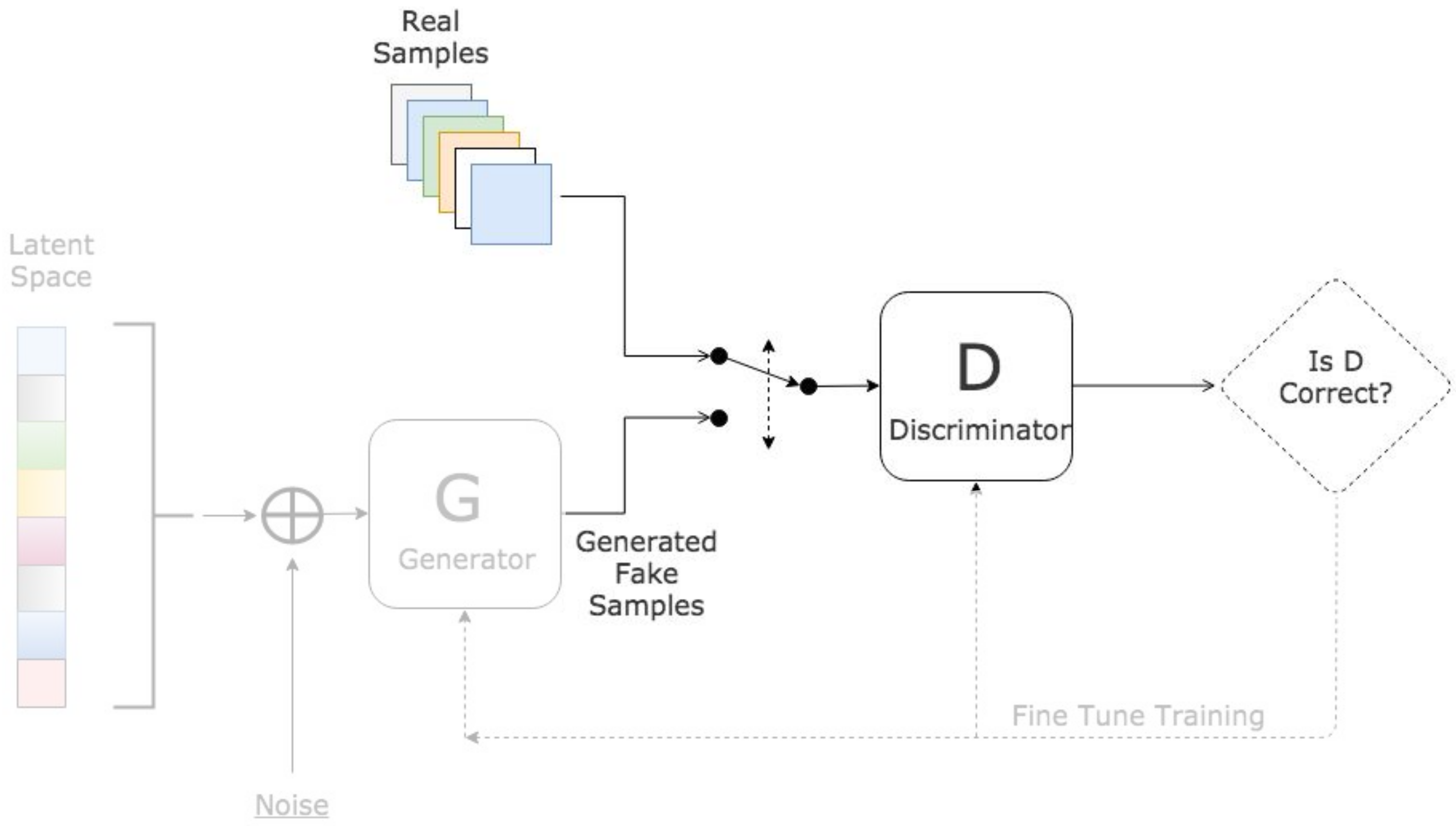


- *The total loss is written as the difference between the generator and the discriminator loss:*
  - *If the discriminator improves, the loss increases*
  - *If the generator improves, the loss decreases*
- *The training continues until the generator fools the discriminator*

European Research Council

Generative Adversarial Training

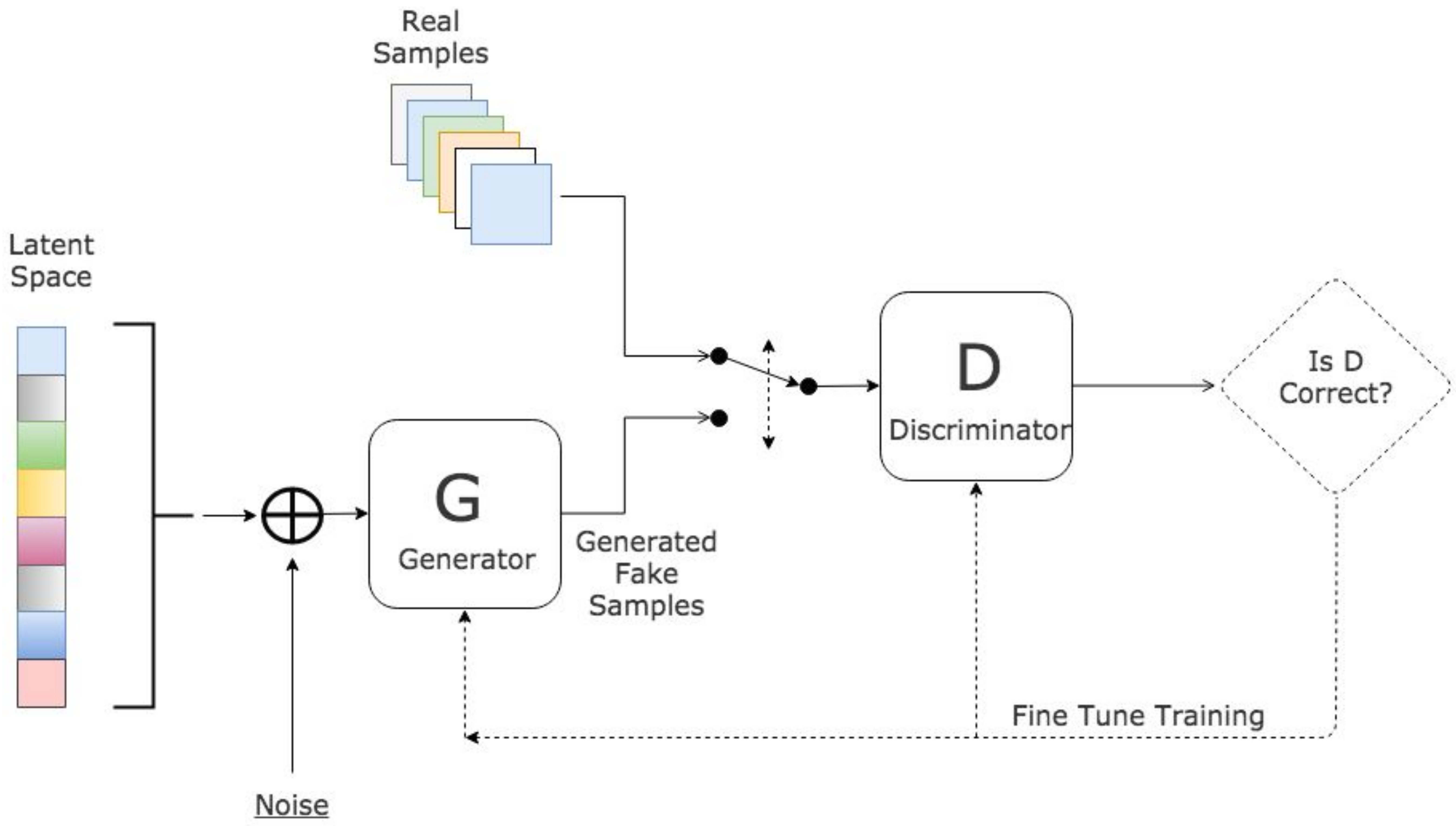
- *Two networks trained against each other*
  - *A generator aims at creating realistic data (e.g., images similar to those in the training dataset)*
  - ***A discriminator aims at identifying which data in a dataset are real and which come from the generator***



- *The total loss is written as the difference between the generator and the discriminator loss:*
  - *If the discriminator improves, the loss increases*
  - *If the generator improves, the loss decreases*
- *The training continues until the generator fools the discriminator*

Generative Adversarial Training

- *Two networks trained against each other*
  - *A generator aims at creating realistic data (e.g., images similar to those in the training dataset)*
  - *A discriminator aims at identifying which data in a dataset are real and which come from the generator*



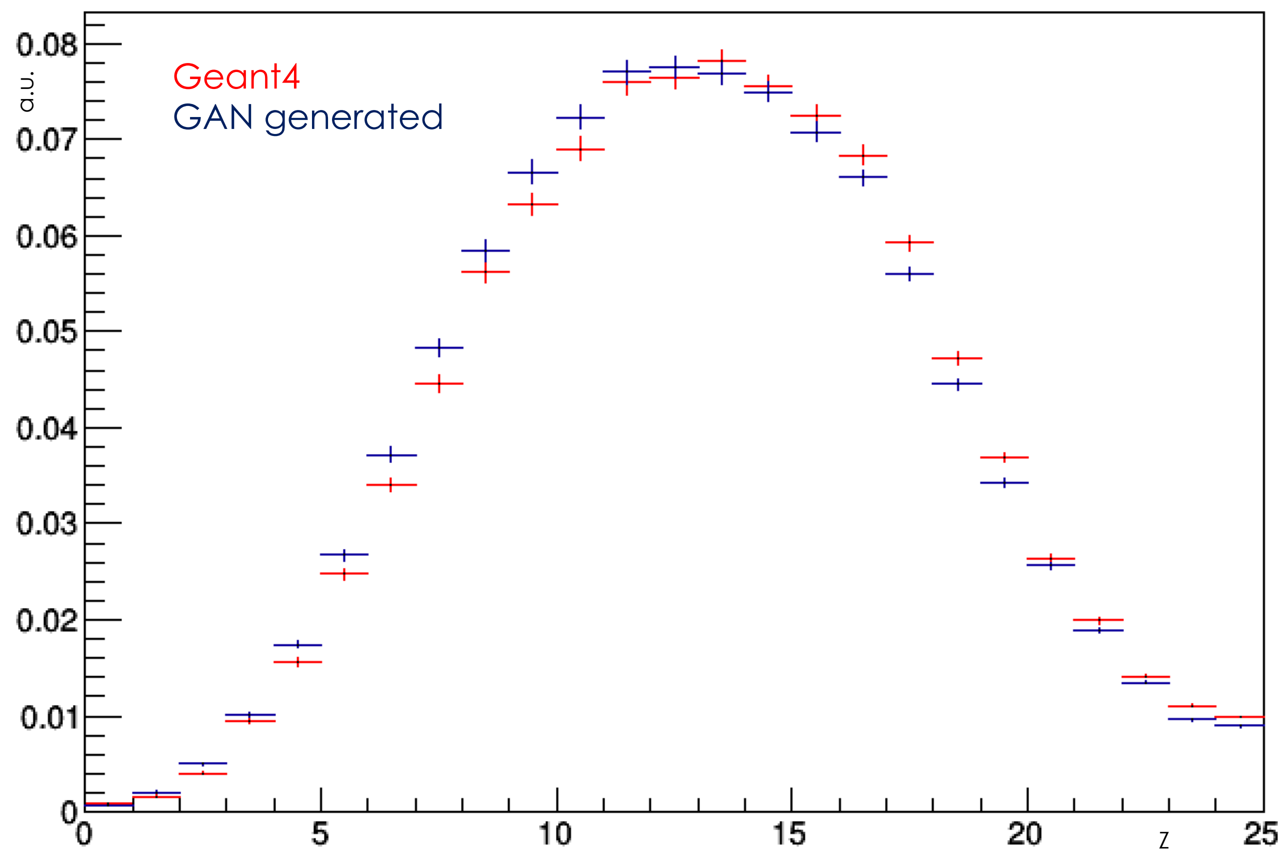
- ***The total loss is written as the difference between the generator and the discriminator loss:***
  - ***If the discriminator improves, the loss increases***
  - ***If the generator improves, the loss decreases***
- ***The training continues until the generator fools the discriminator***

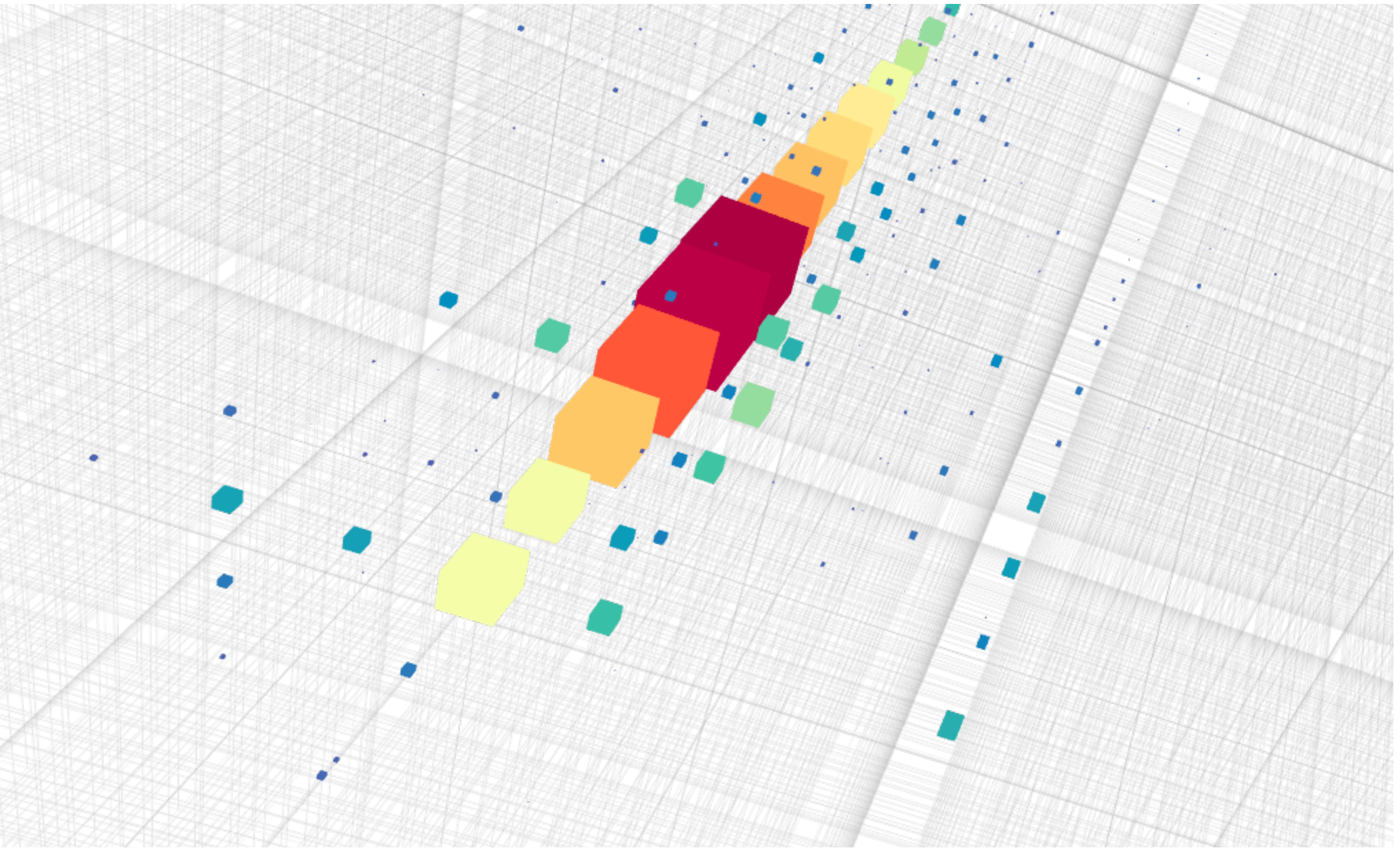
Generative Adversarial training in action

Generating detector response

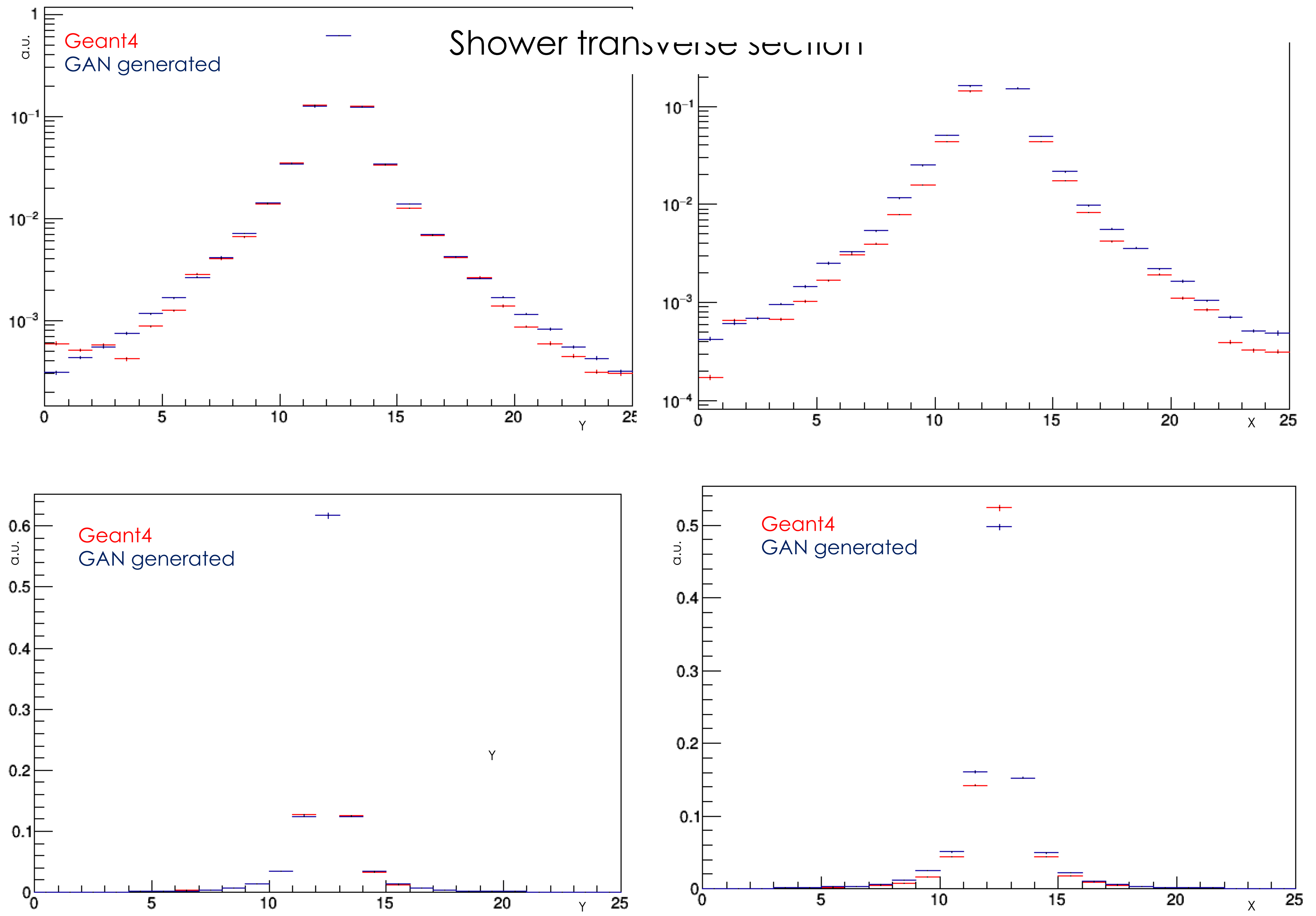
- Start from random noise
- Works very well with images
  - Applied to electron showers in digital calorimeters as a replacement of GEANT

Shower longitudinal section





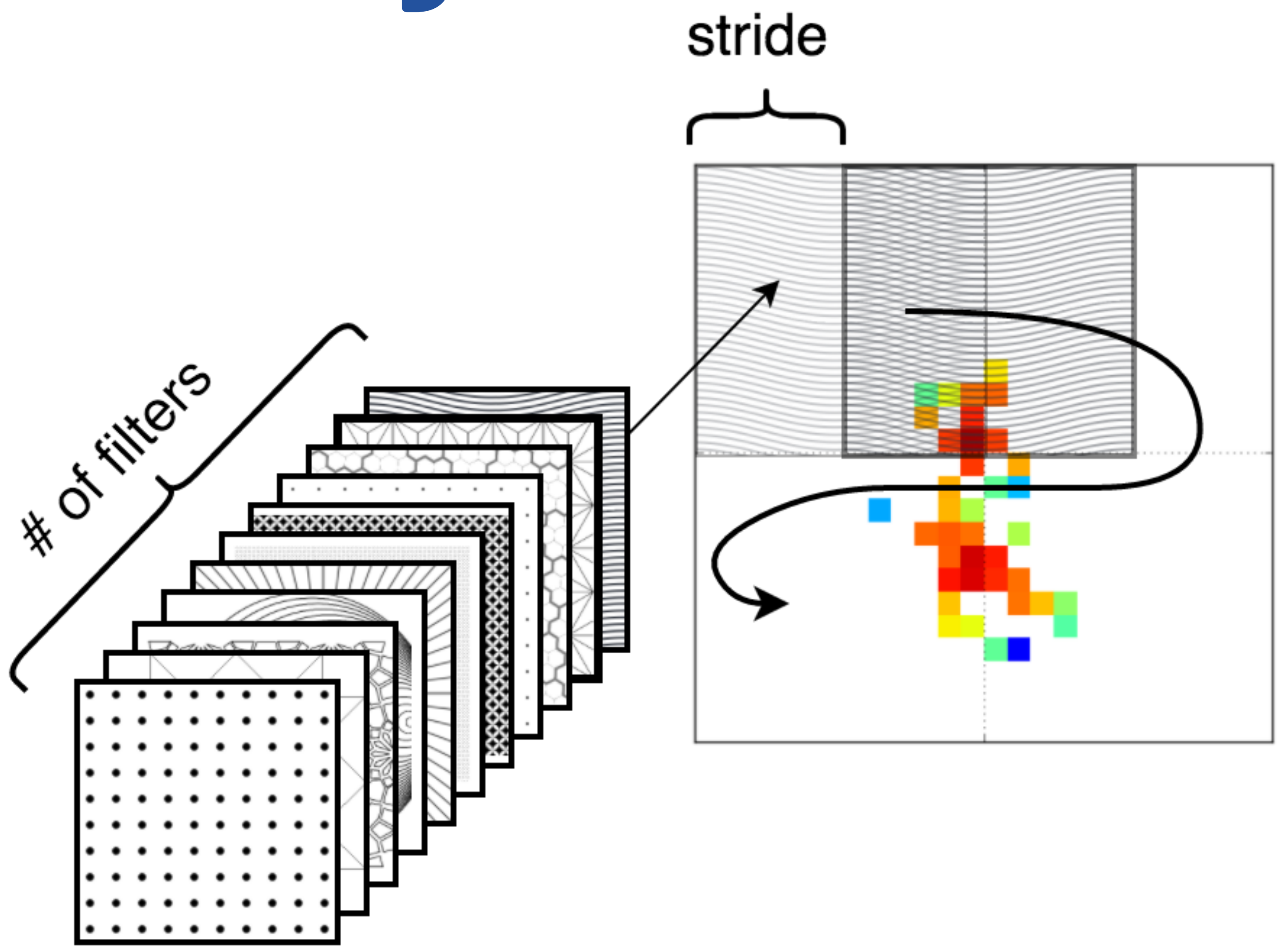
Shower transverse section



[See contribution to NIPS workshop](#)

Generating a full jets

- Start from random noise
- Works very well with images
  - Applied to electron showers in digital calorimeters as a replacement of GEANT



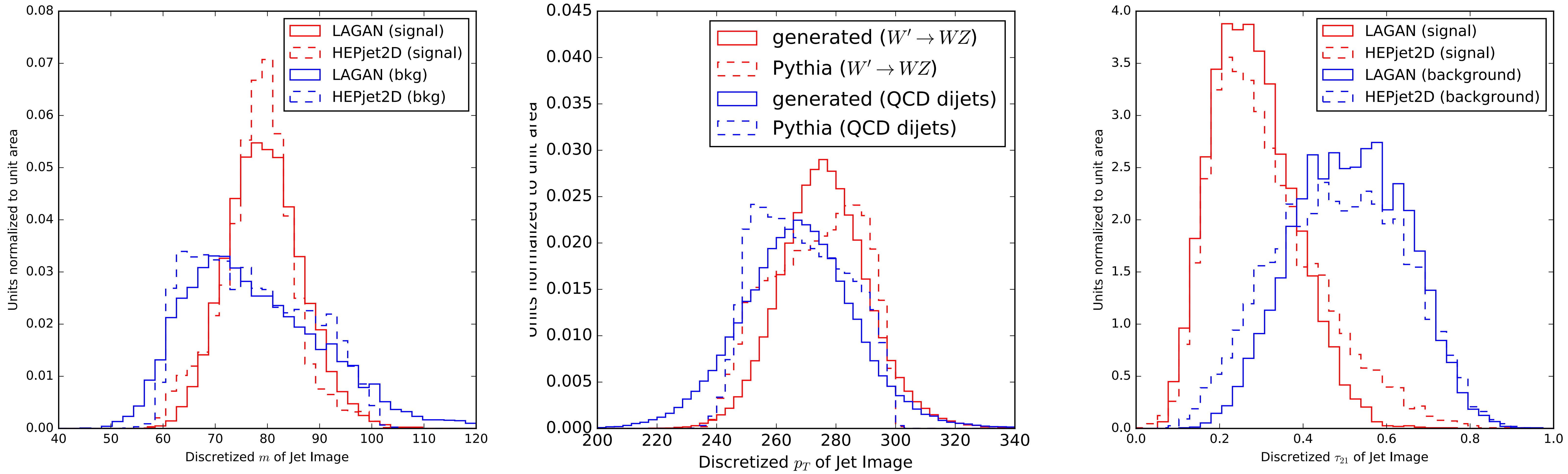
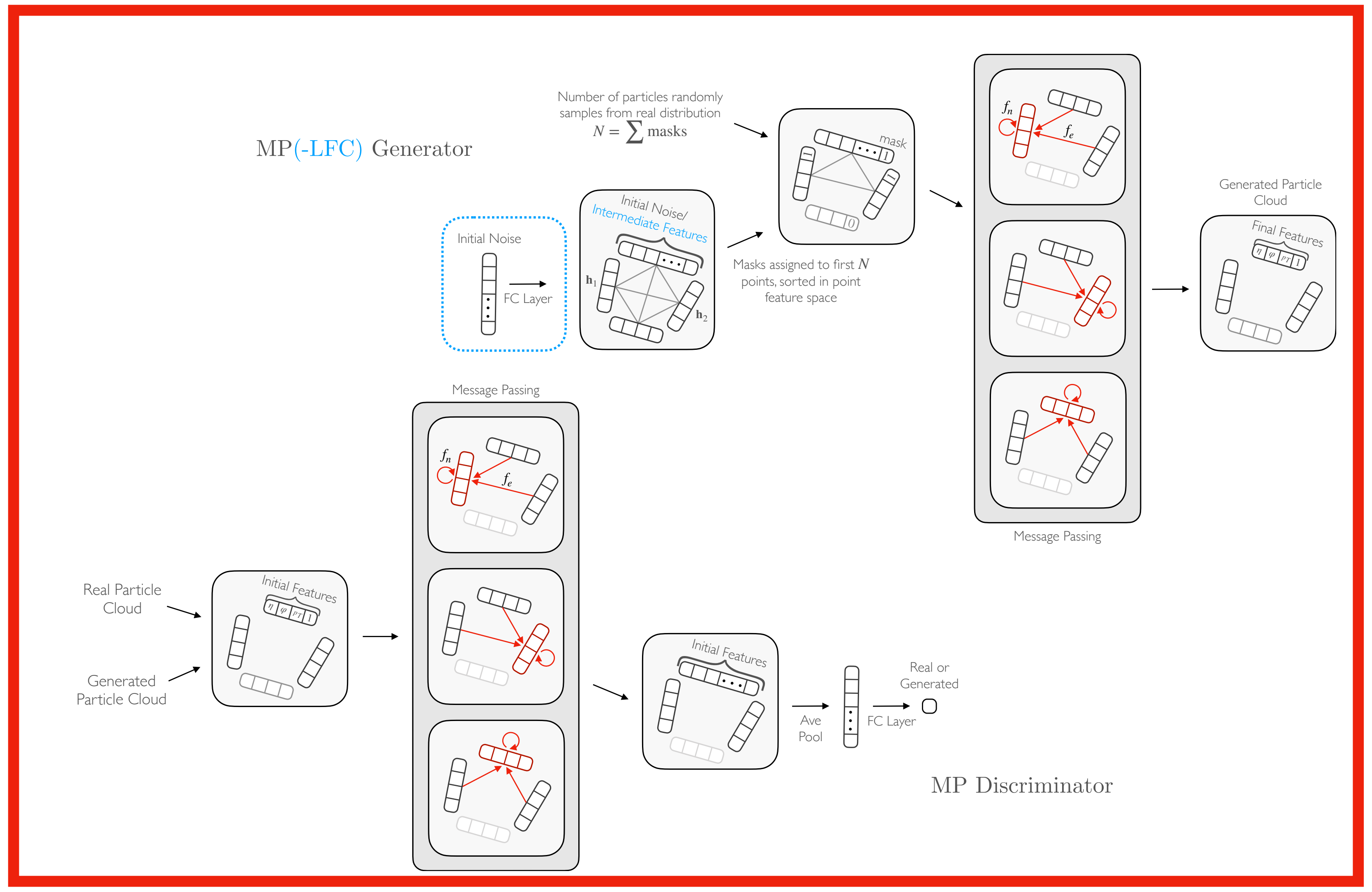


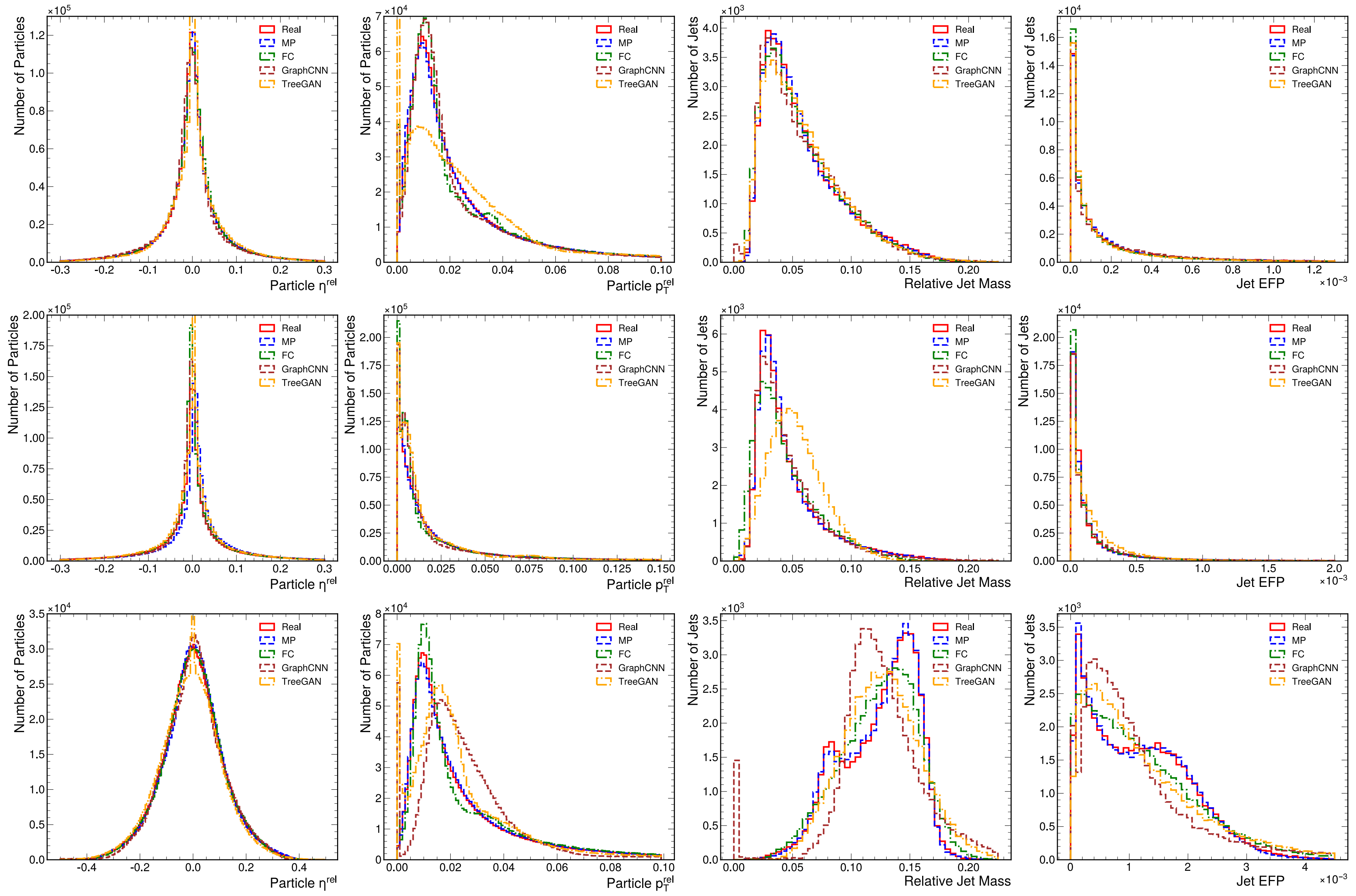
Figure 6: The distributions of image mass $m(I)$, transverse momentum $p_{\mathrm{T}}(I)$, and n-subjettiness $\tau_{21}(I)$. See the text for definitions.

de Olivera, Paganini, and Nachman
https://arxiv.org/pdf/1701.05927.pdf

Same problems, same solution

- *As for reconstruction, the ultimate challenge of DL for simulation is the sparse nature of the data*
- *As for reconstruction, a solution is adopting Graph Architectures*
- *Graph GANs have been successfully trained (e.g., to reconstruct jets)*
- *Work ongoing to scale up the models, so that graphs of O(1000) could ge generated*

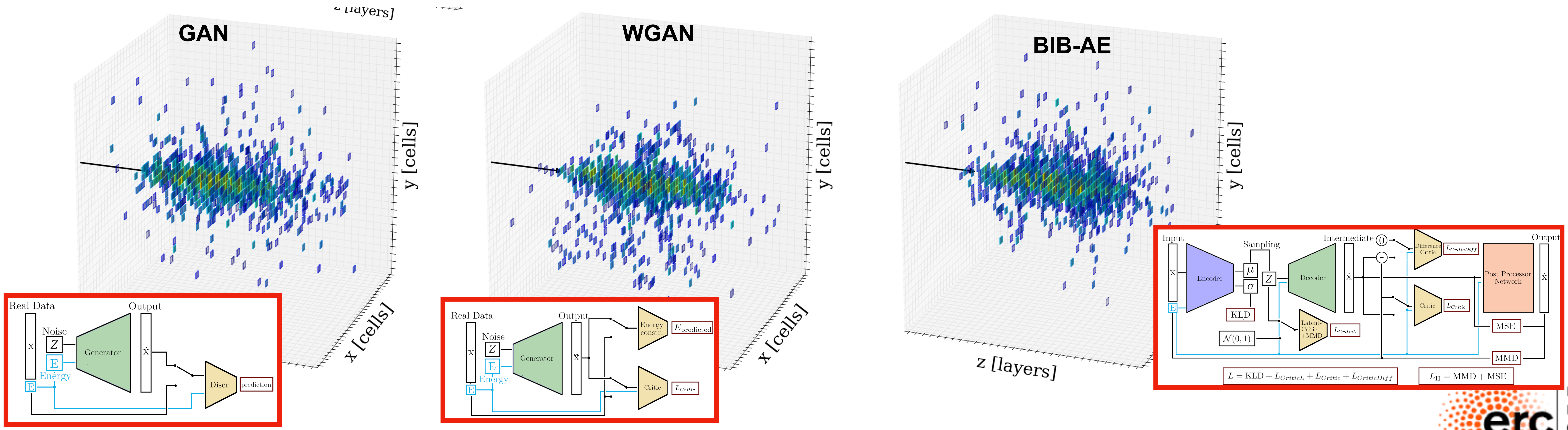




Kansal et al. https://arxiv.org/pdf/2106.11535.pdf

Same problems, same solution

- *As for reconstruction, the ultimate challenge of DL for simulation is the sparse nature of the data*
- *As for reconstruction, a solution is adopting Graph Architectures*
- *Graph GANs have been successfully trained (e.g., to reconstruct jets)*
- *Work ongoing to scale up the models, so that graphs of O(1000) could ge generated*



Buhmann et al. https://arxiv.org/pdf/2005.05334.pdf

European Research Council

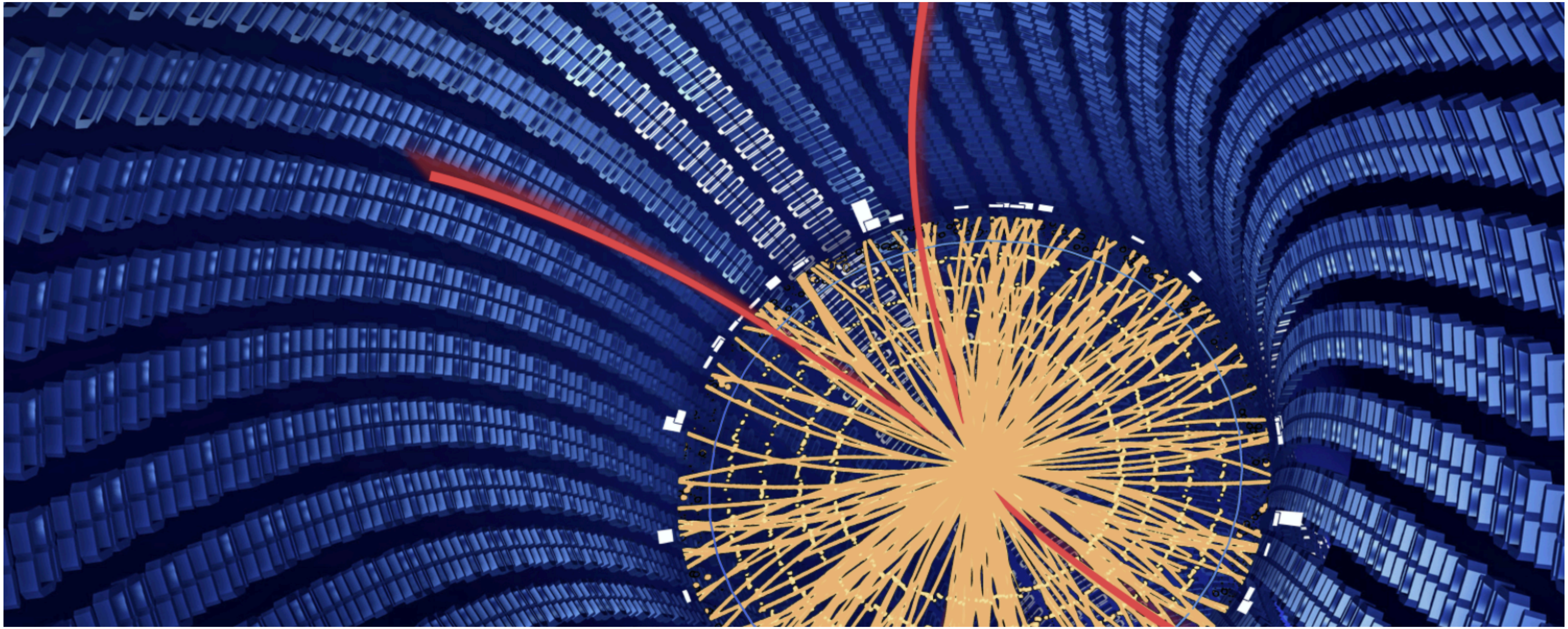
Summary of Lecture 2

- We looked into two applications of Neural Networks
  - Reconstruction of particles in LHC detector from the "hits" left by particles generated in the collision
  - Simulation of the hits left by the particles generated in the collision
- Both problems require ones to deal with the sparse and irregular nature of the data
  - Particle physics data are point clouds
  - Graph neural networks can effectively solve problems with point-cloud data

Further Reading & Coding

- *A few recent reviews that could guide you through the many applications and networks*
  - *A nice BLOG article on GNNs*
  - *Another nice BLOG article on GNNs*
  - *A generic review*
  - *A particle-physics specific one*
- *And the study from which our hands-on session comes*
  - *JEDI-net Interaction Networks for jet tagging on these data*

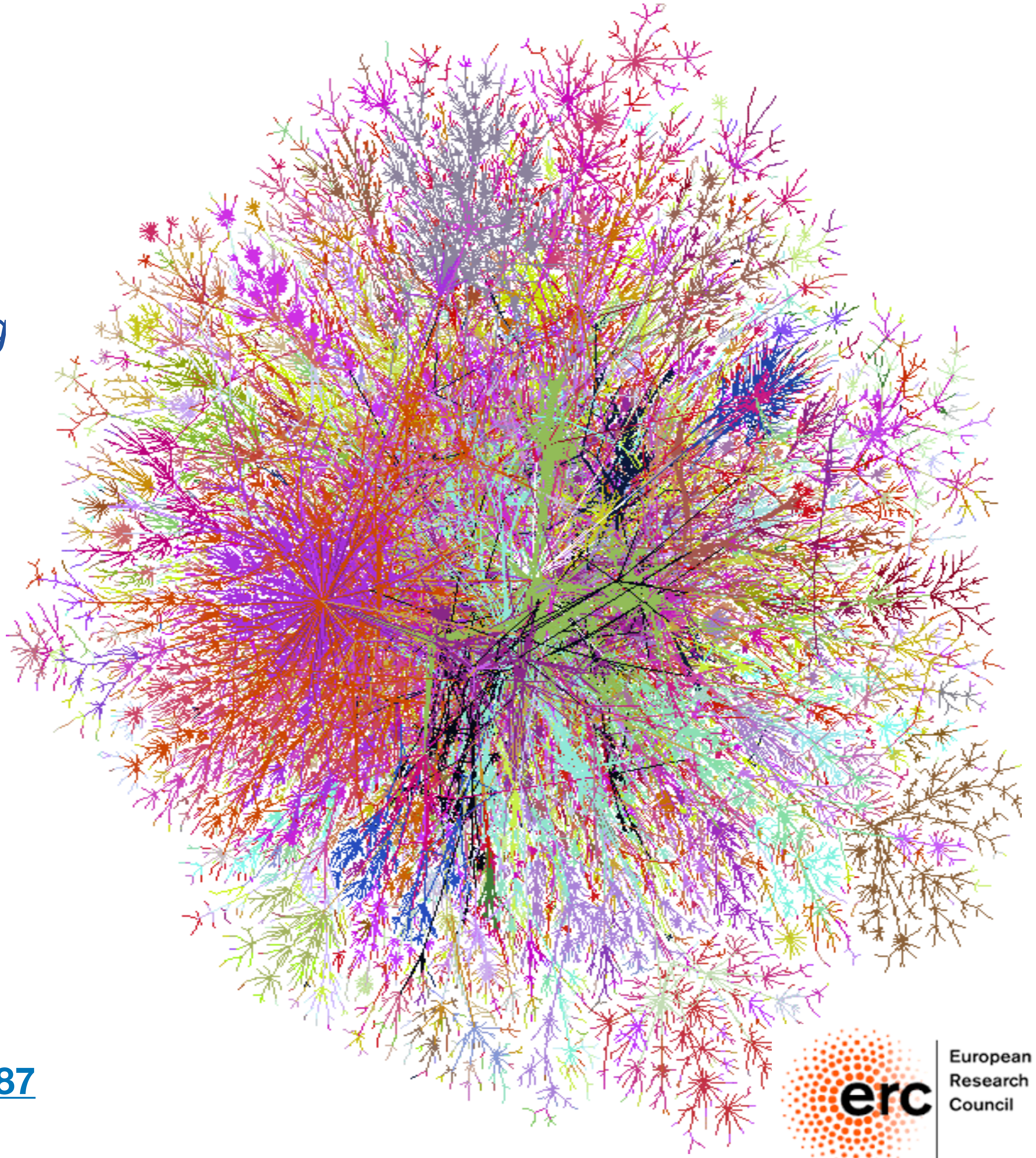
European Research Council



Backup

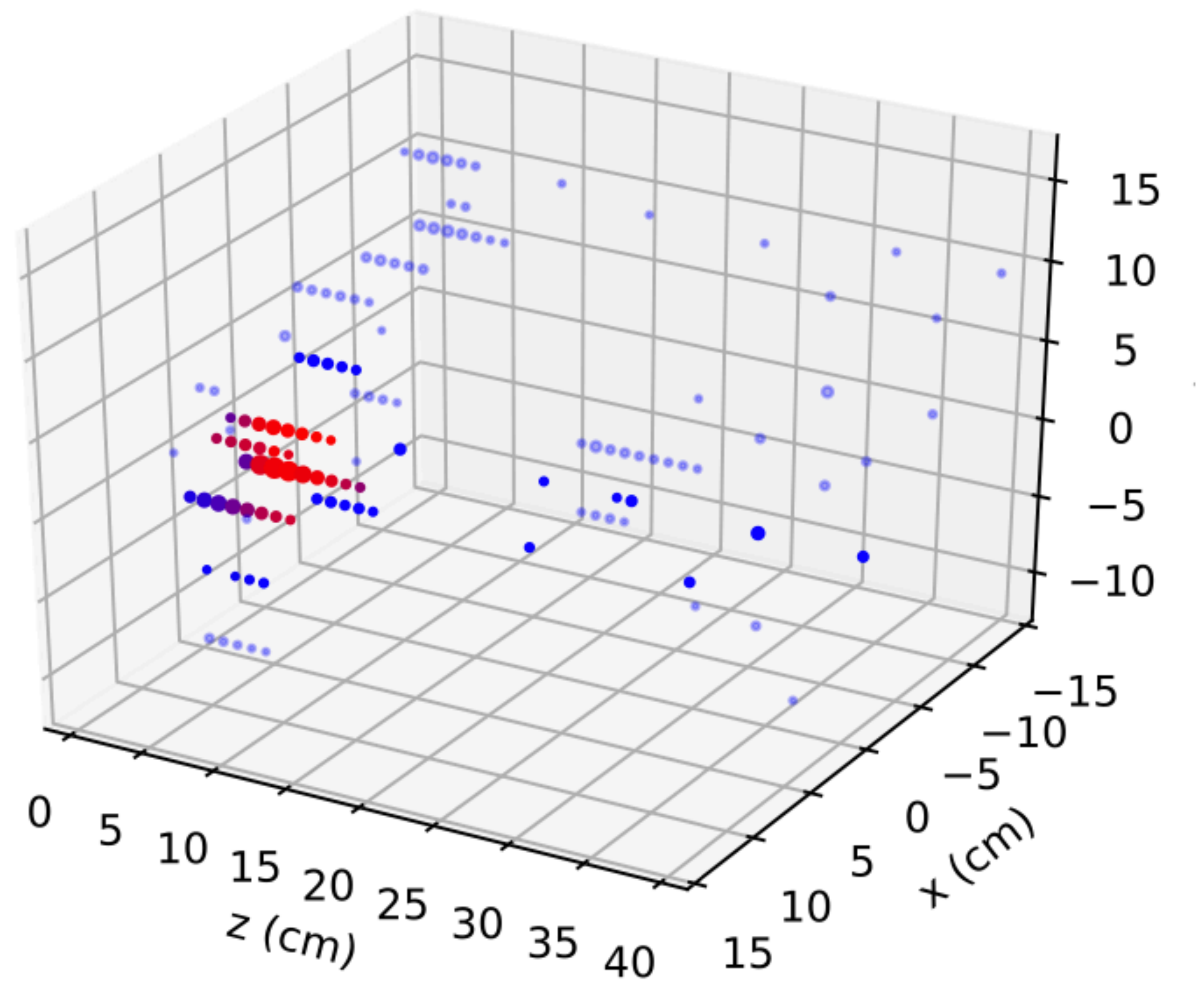
Reducing memory consumption

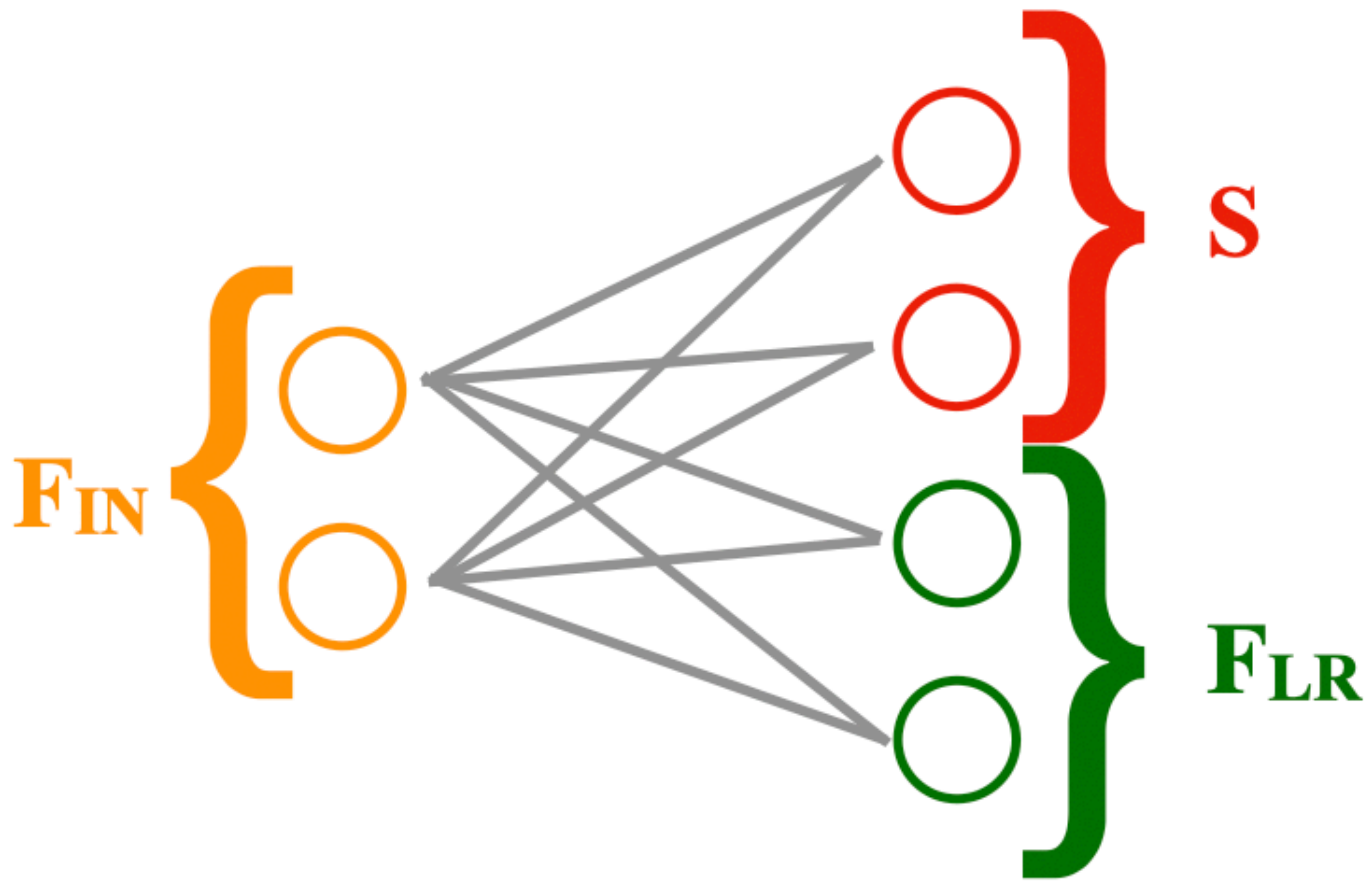
- *When building a graph of N vertices, number of edges (and number of computing operations) scale with N^2*
- *This might clash with computing resource limitations (both for training and inference)*
- *Certainly, this is the case at the LHC*
    - *real-time event selection runs in short time*
    - *most of the selection runs as electronic circuit on electronic board*
- *Gravnet & Garnet: resource friendly graph architectures* https://arxiv.org/abs/1902.07987

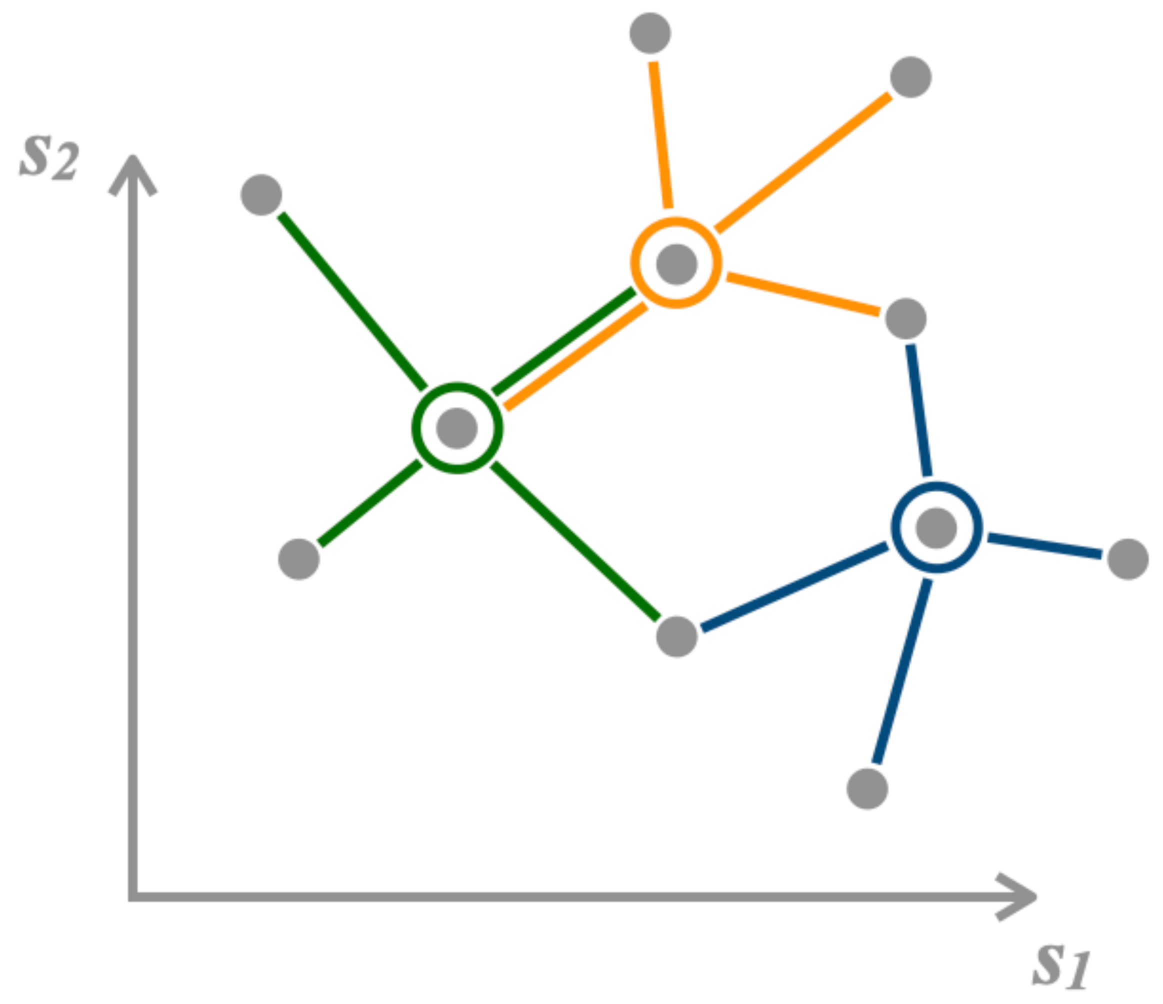


GravNet

https://arxiv.org/abs/1902.07987







1) Start with a graph in geometric space. Each vertex feature vector F_{IN} is characterized by coordinates and features

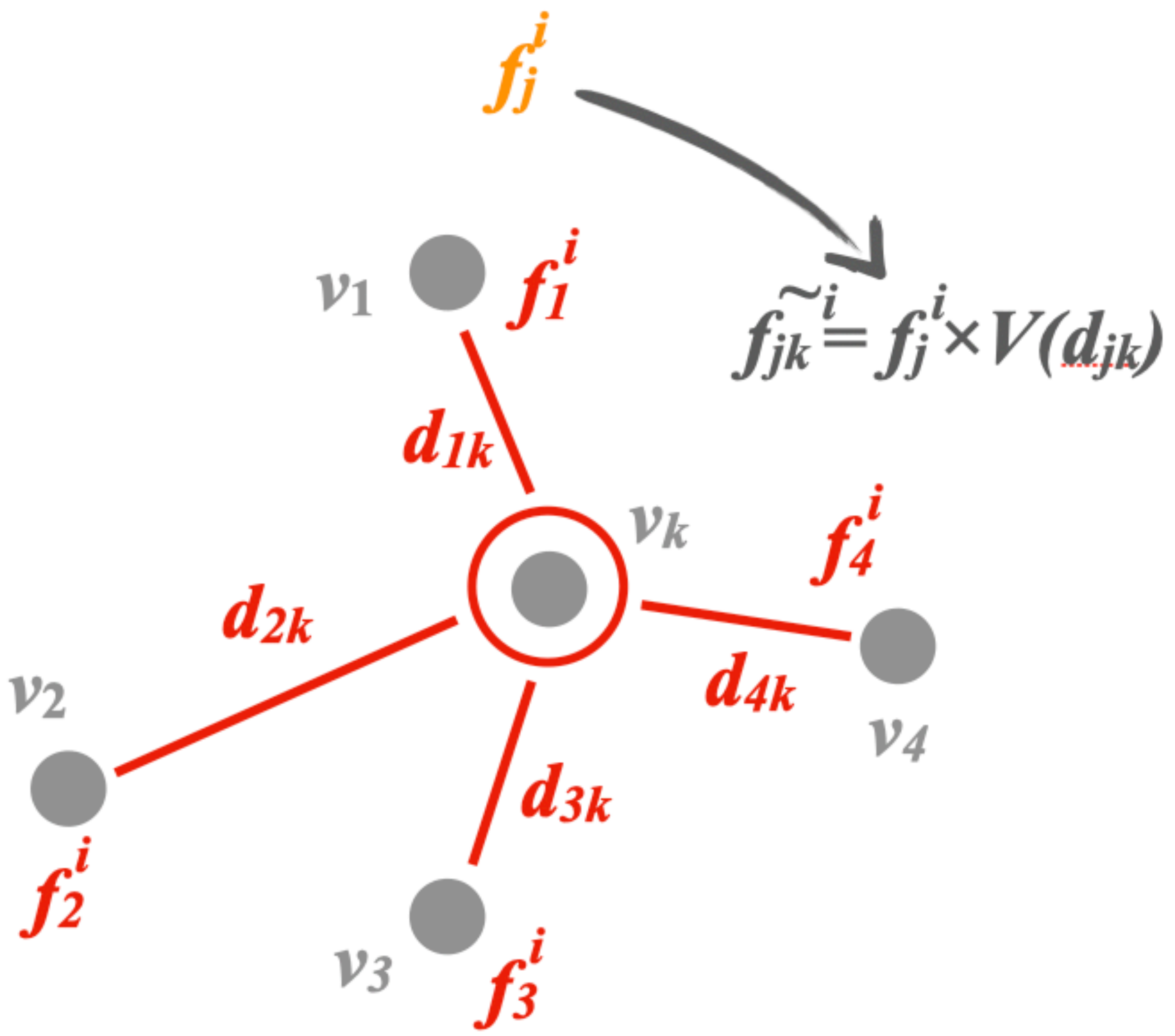
2) Each F_{IN} is processed by a linear network, returning two outputs: a coordinate vector s & a learned representation F_{LR}

3) With s and F_{LR} we build the new graph in the learned space

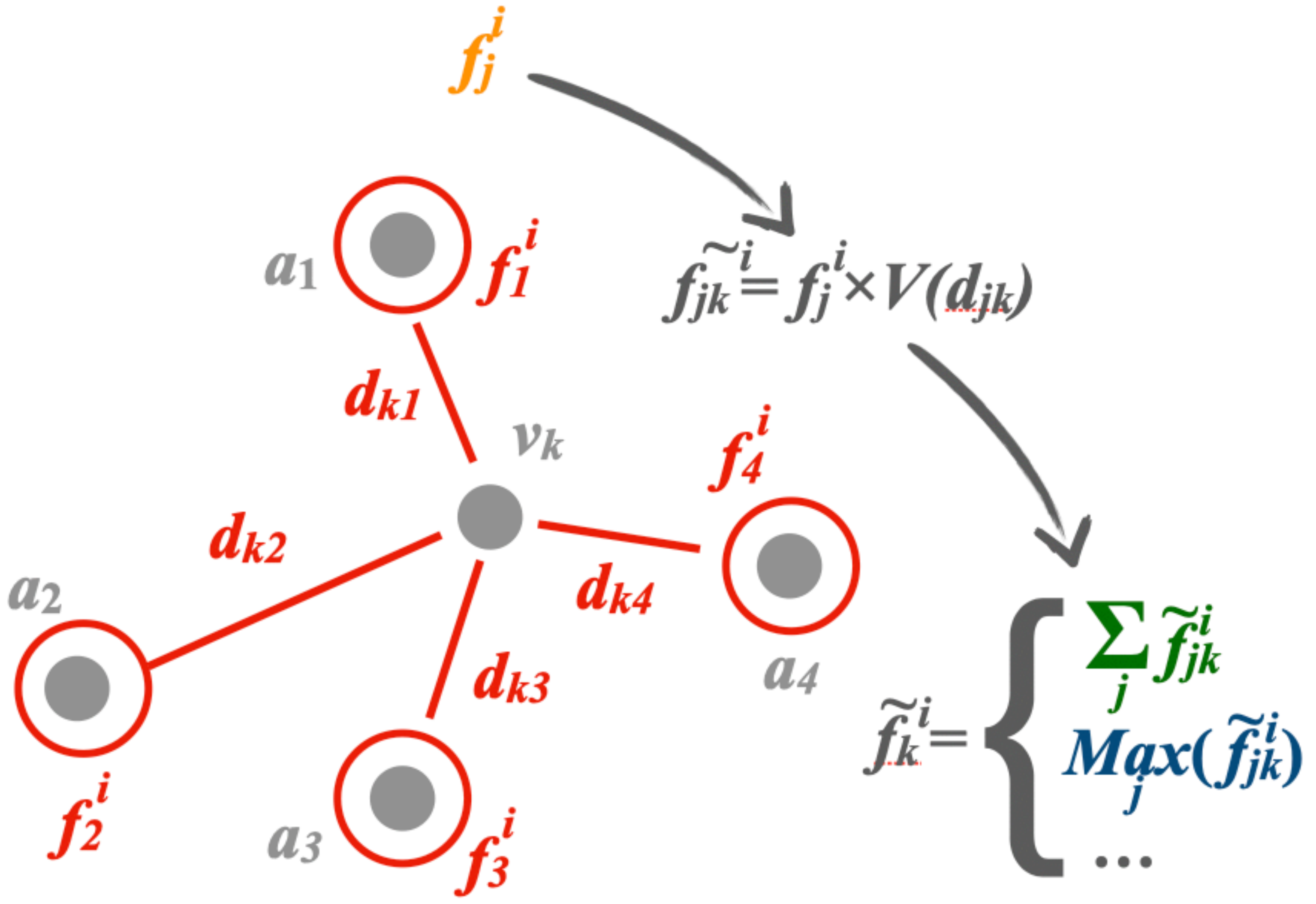
European Research Council

GravNet

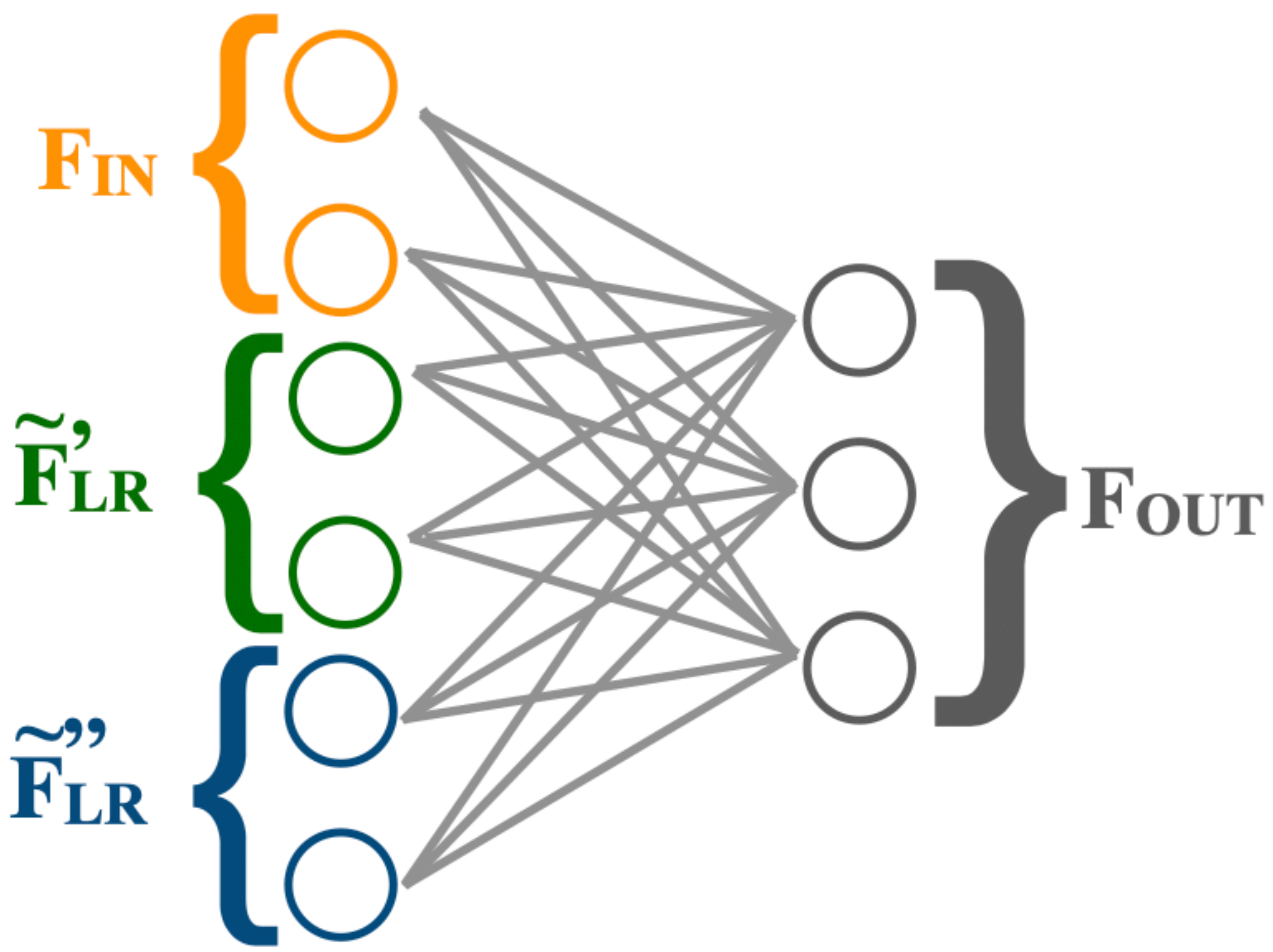
https://arxiv.org/abs/1902.07987



4) Unlike DGCNN, the message function is a potential function (we use e^{-d^2} where d is the Euclidean distance in learned space)

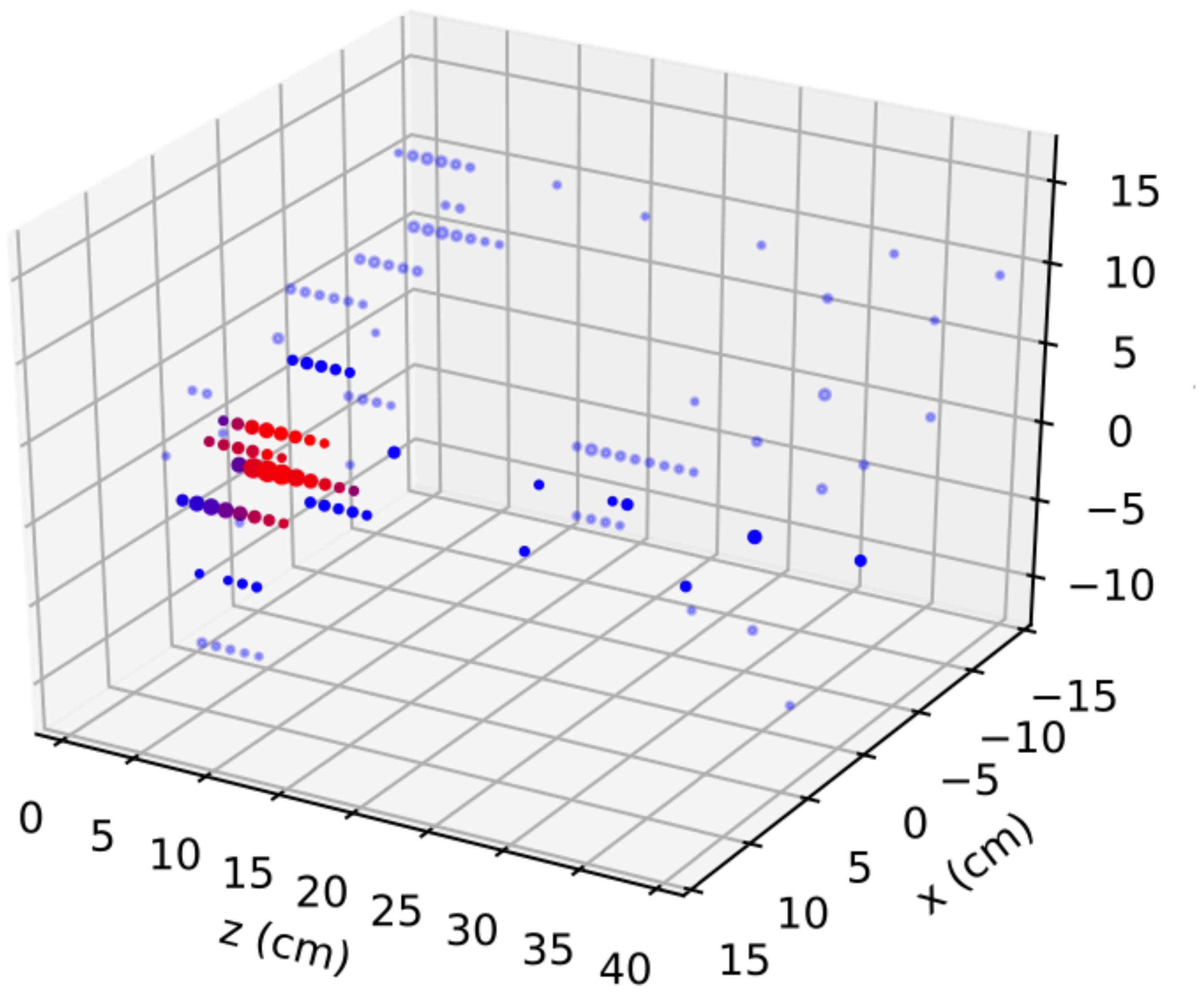


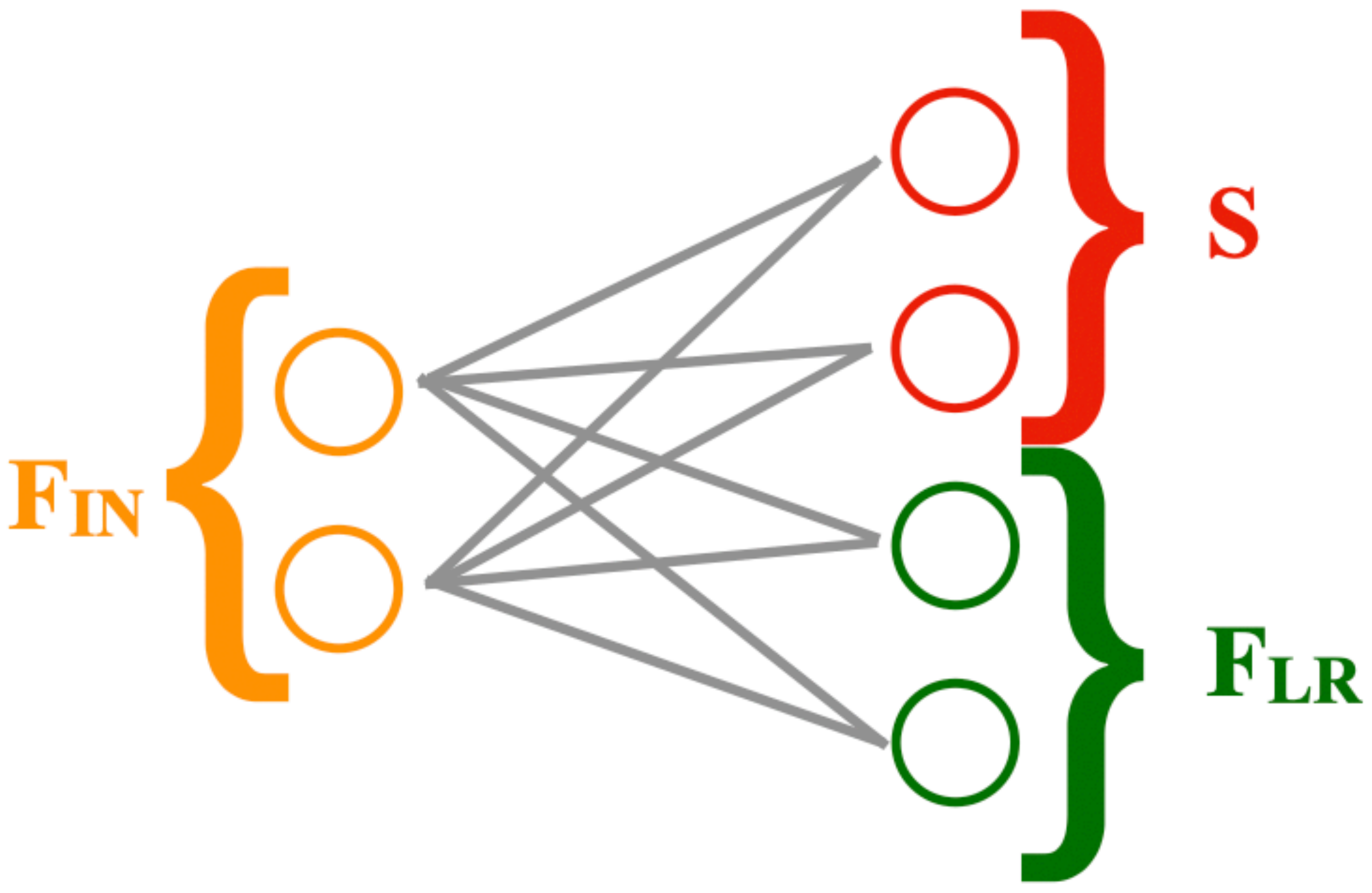
5) Message aggregated with different functions (Max, Average,…)

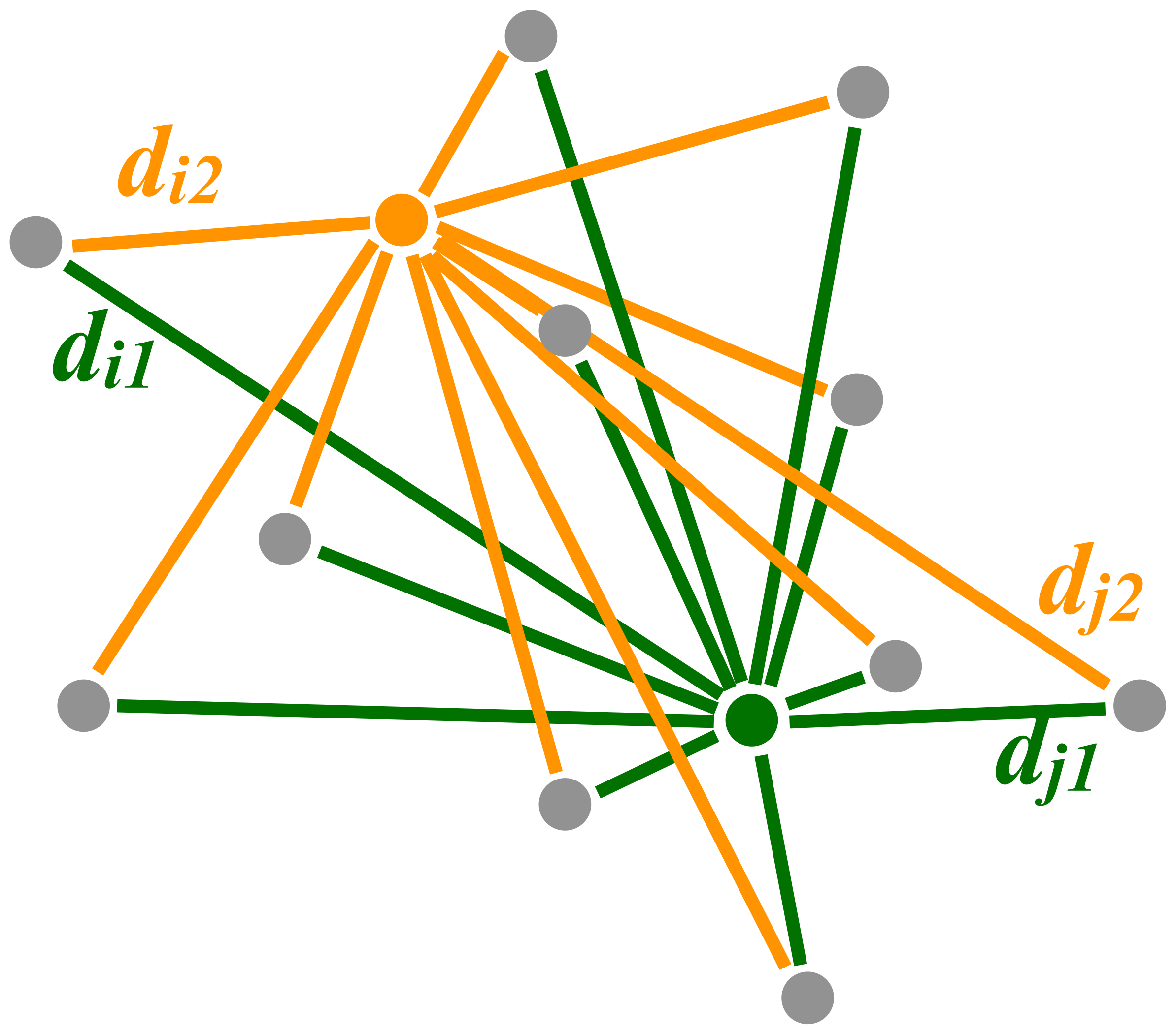


6) Final representation is learned from the engineered features and the original ones

(simplified) GarNet







1) Start with a graph in geometric space. Each vertex feature vector F_{IN} is characterized by coordinates and features

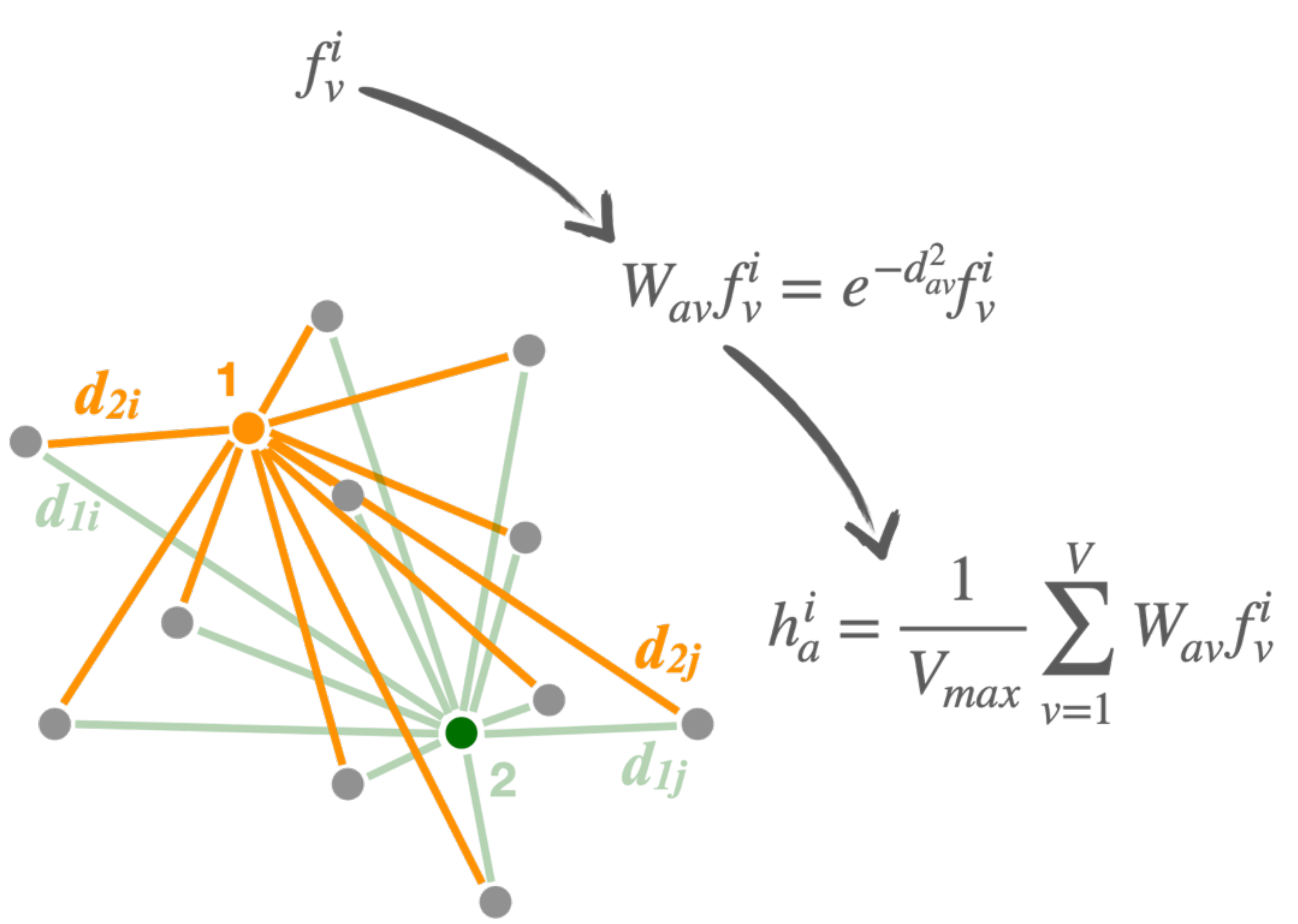
2) Each F_{IN} is processed by a linear network, returning two outputs: a vector of distances s & a learned representation F_{LR}

3) s are the distances from Ns aggregators

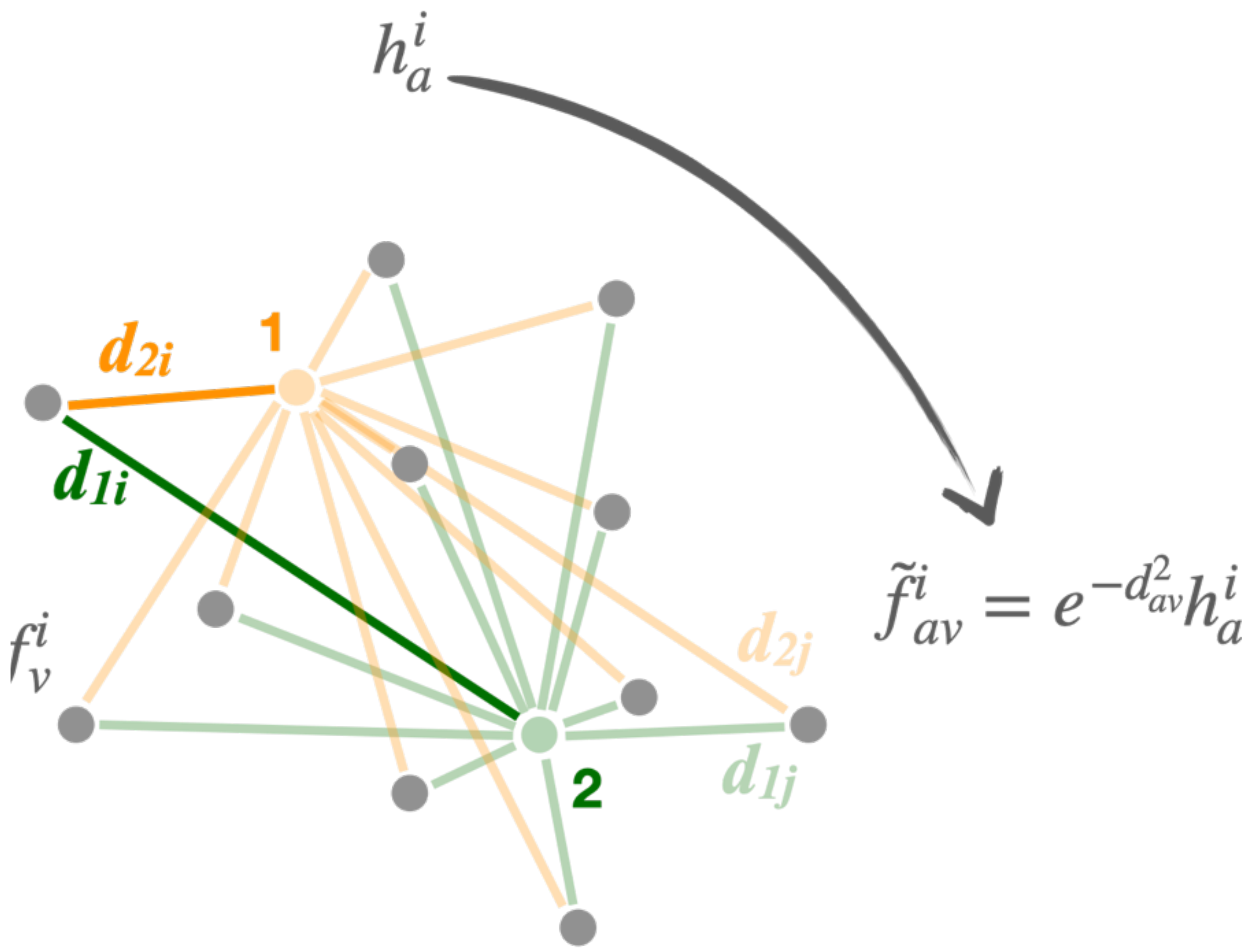
https://arxiv.org/abs/1902.07987

https://arxiv.org/pdf/2008.03601.pdf

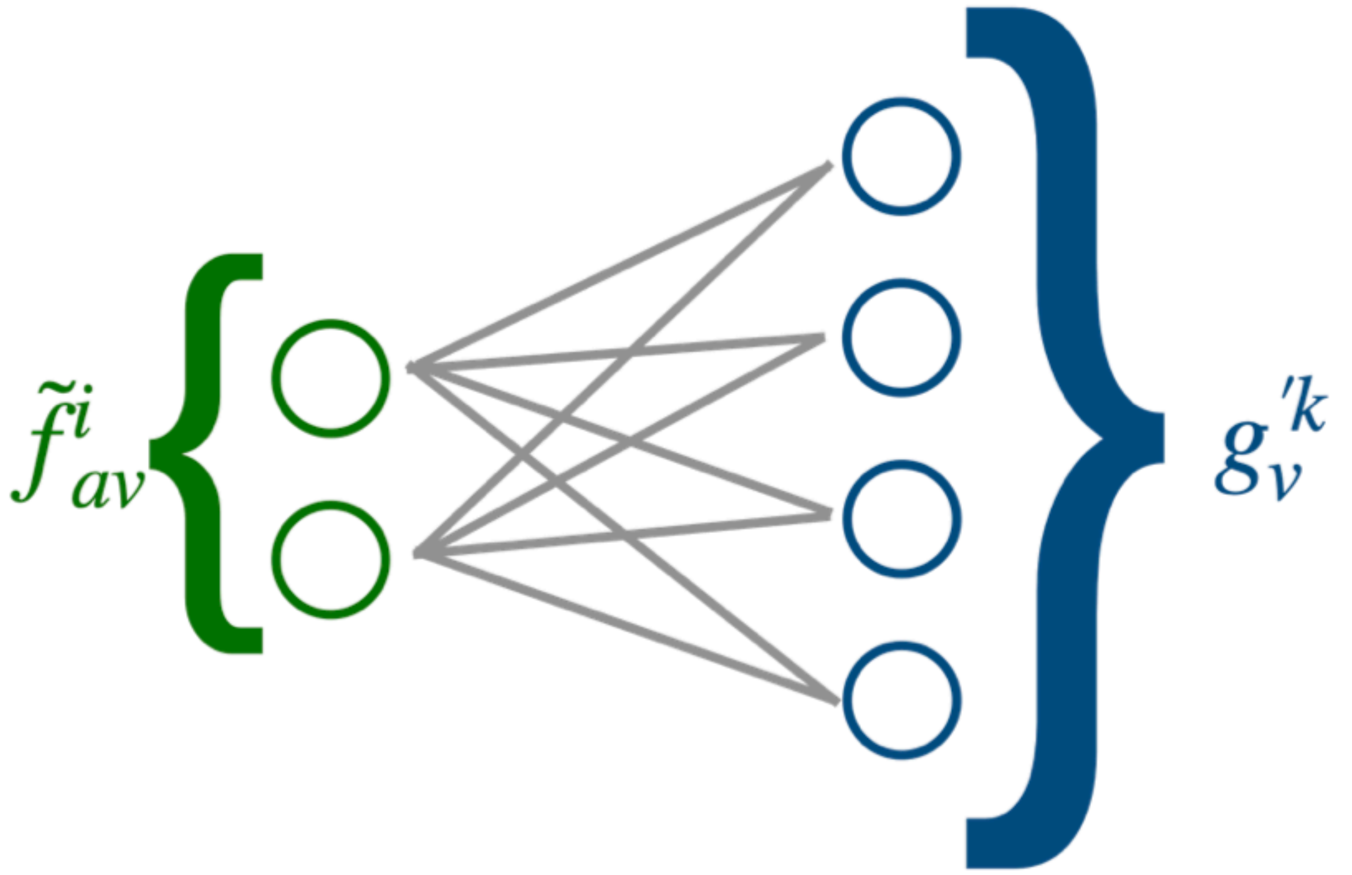
(simplified) GarNet



4) Fwd distance-weighted messages from vertices are gathered at aggregators (weight $W_{ab} = e^{-d_{ab}}$ where d is Euclidean distance in learned space)



5) Bkw distance-weighted messages from aggregators are gathered at vertices (weight $W_{ab} = e^{-d_{ab}}$)



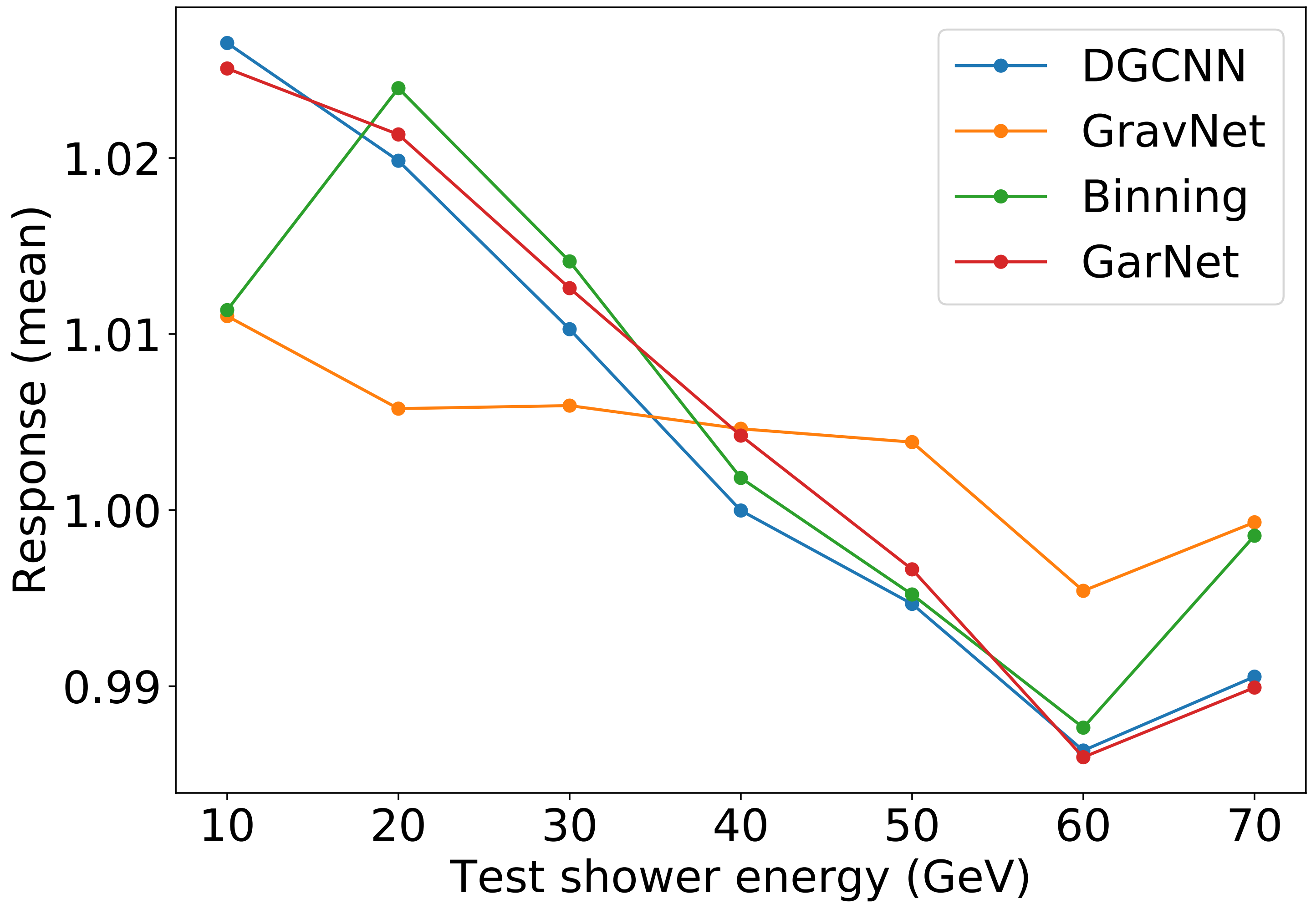
6) Final representation is learned from the engineered features and the original ones

https://arxiv.org/abs/1902.07987

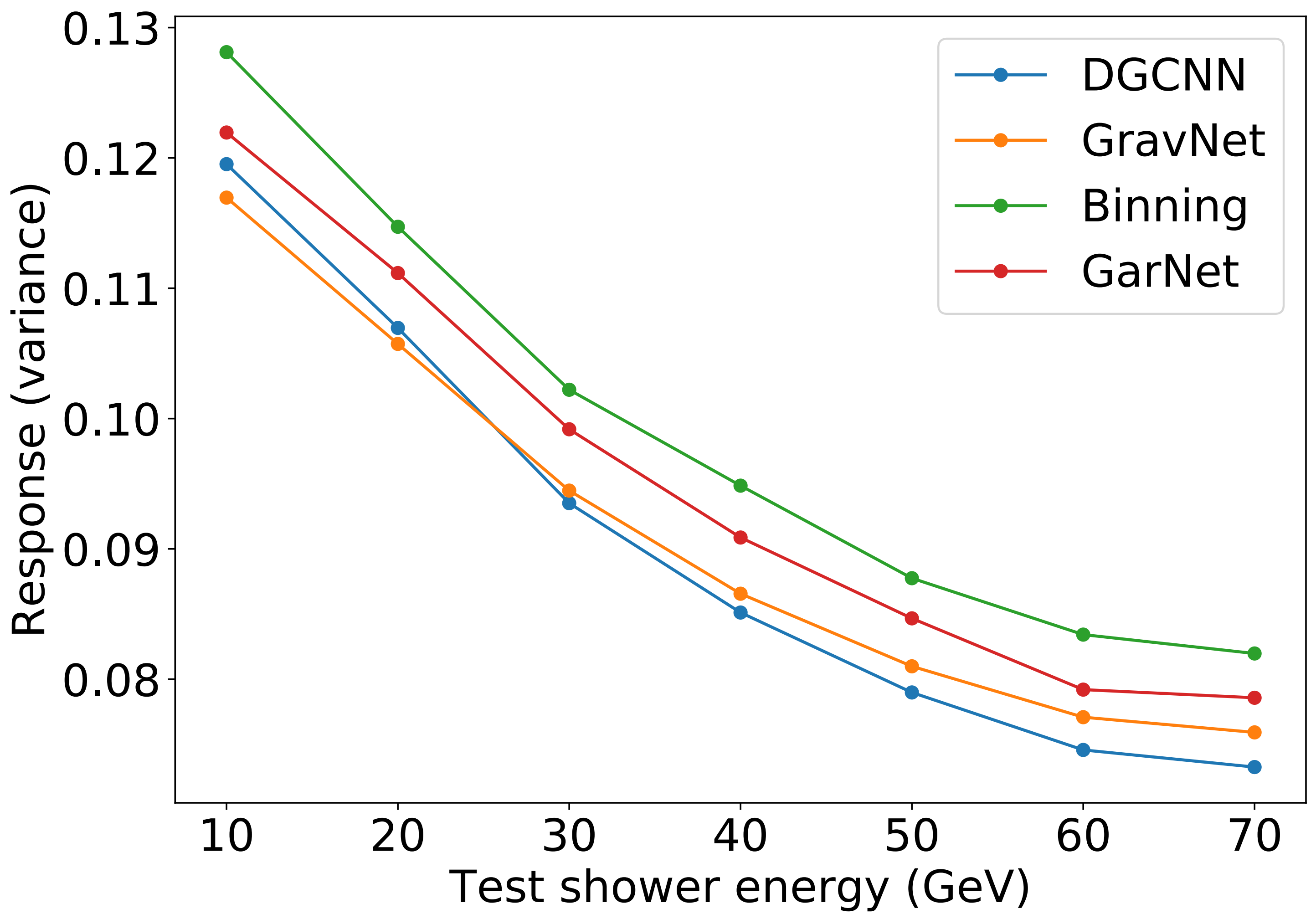
https://arxiv.org/pdf/2008.03601.pdf

GarNet & GravNet for Calorimetry

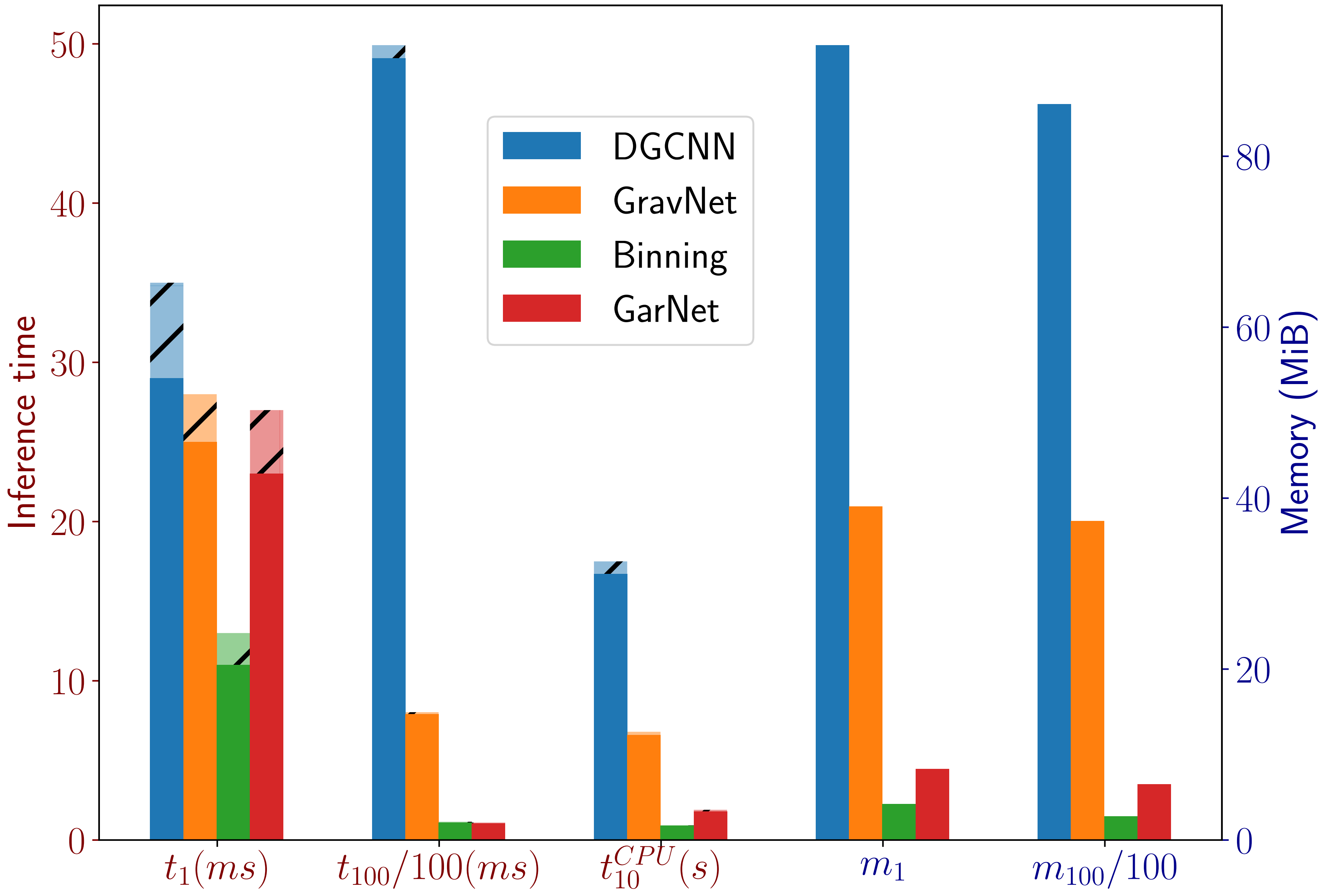
- Good performance achieved, comparable to DGCNN and traditional approaches
- Using a potential (V(d)) to weight up the near neighbours allows to keep memory footprint under control (with respect to other graph approaches)



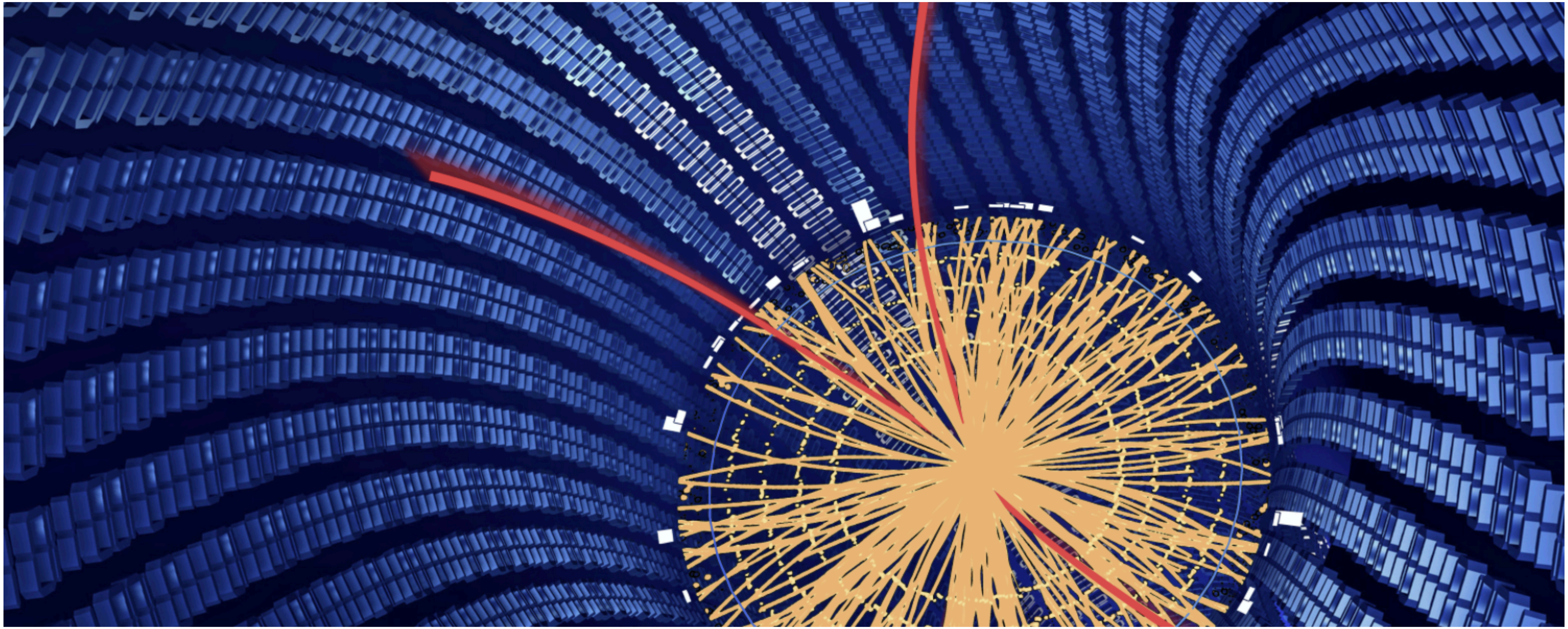
(c) Mean



(d) Variance



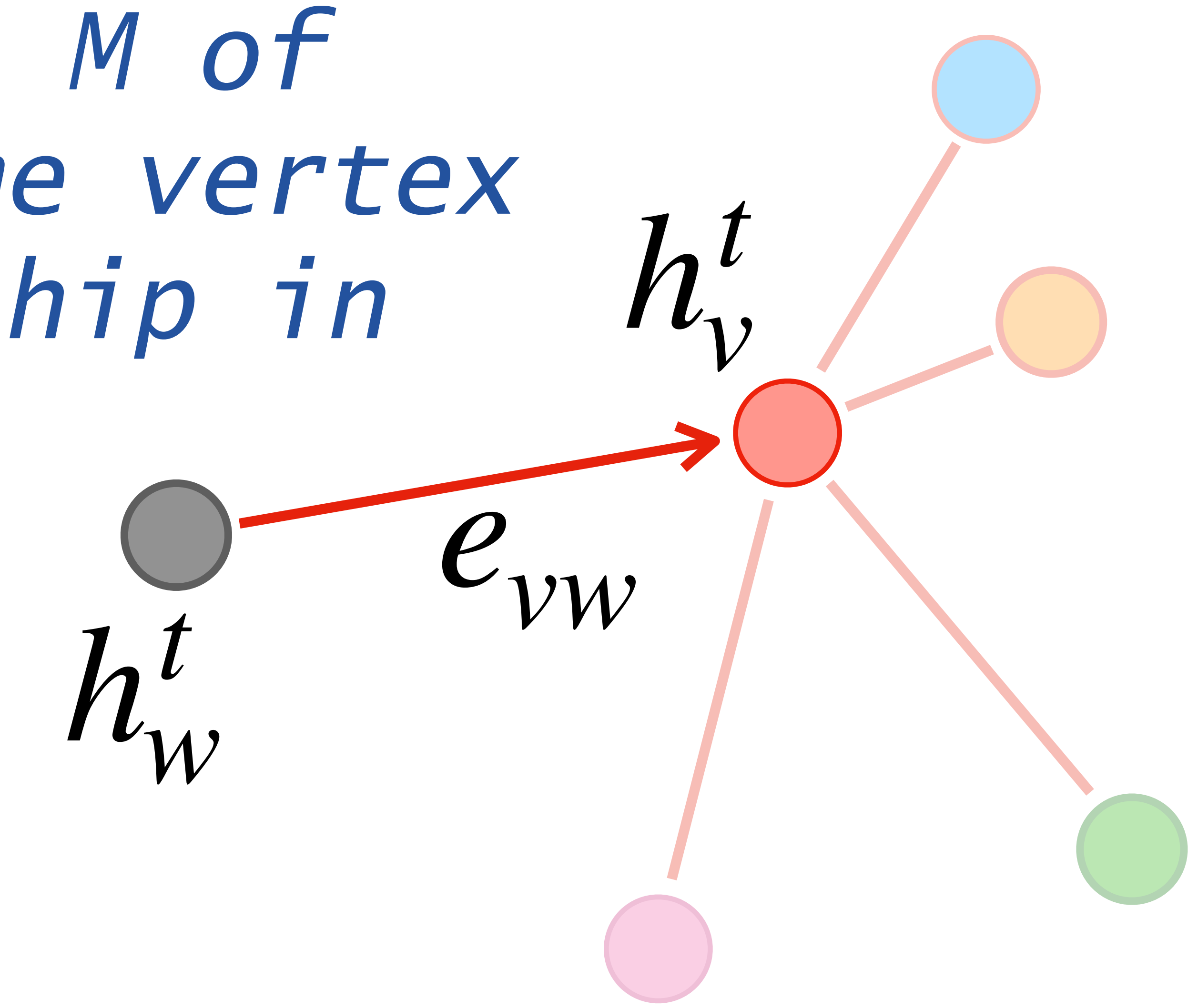
European Research Council



Physics and Deep Learning: more thoughts from Lecture 1

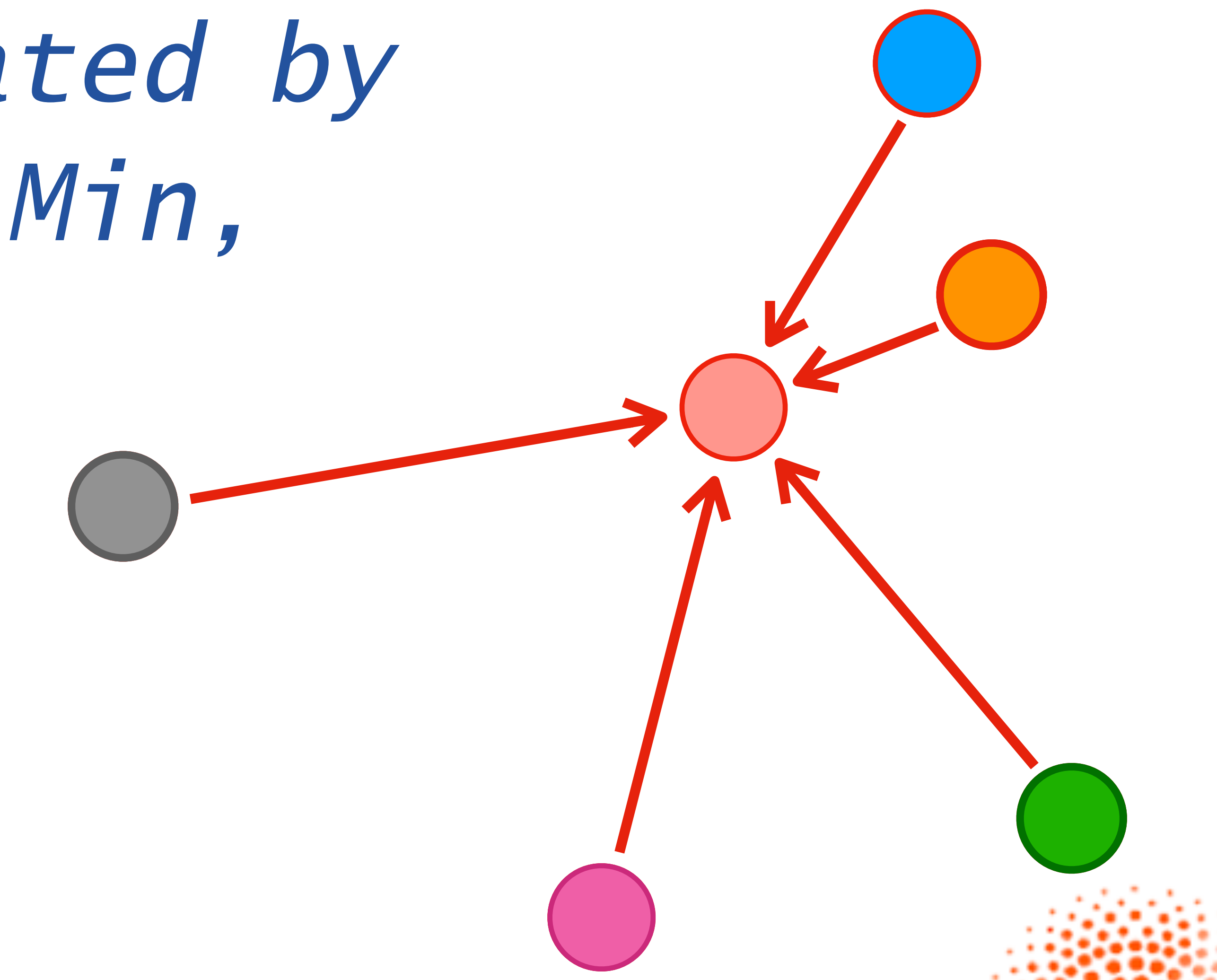
With equations…

- *Your message at iteration t is some function M of the sending and receiving features, plus some vertex features (e.g., business relation vs friendship in social media)*



$$M_t(h_v^t, h_w^t, e_{vw})$$

- *The message carried to a vertex v is aggregated by some function (typically sum, but also Max, Min, etc.)*

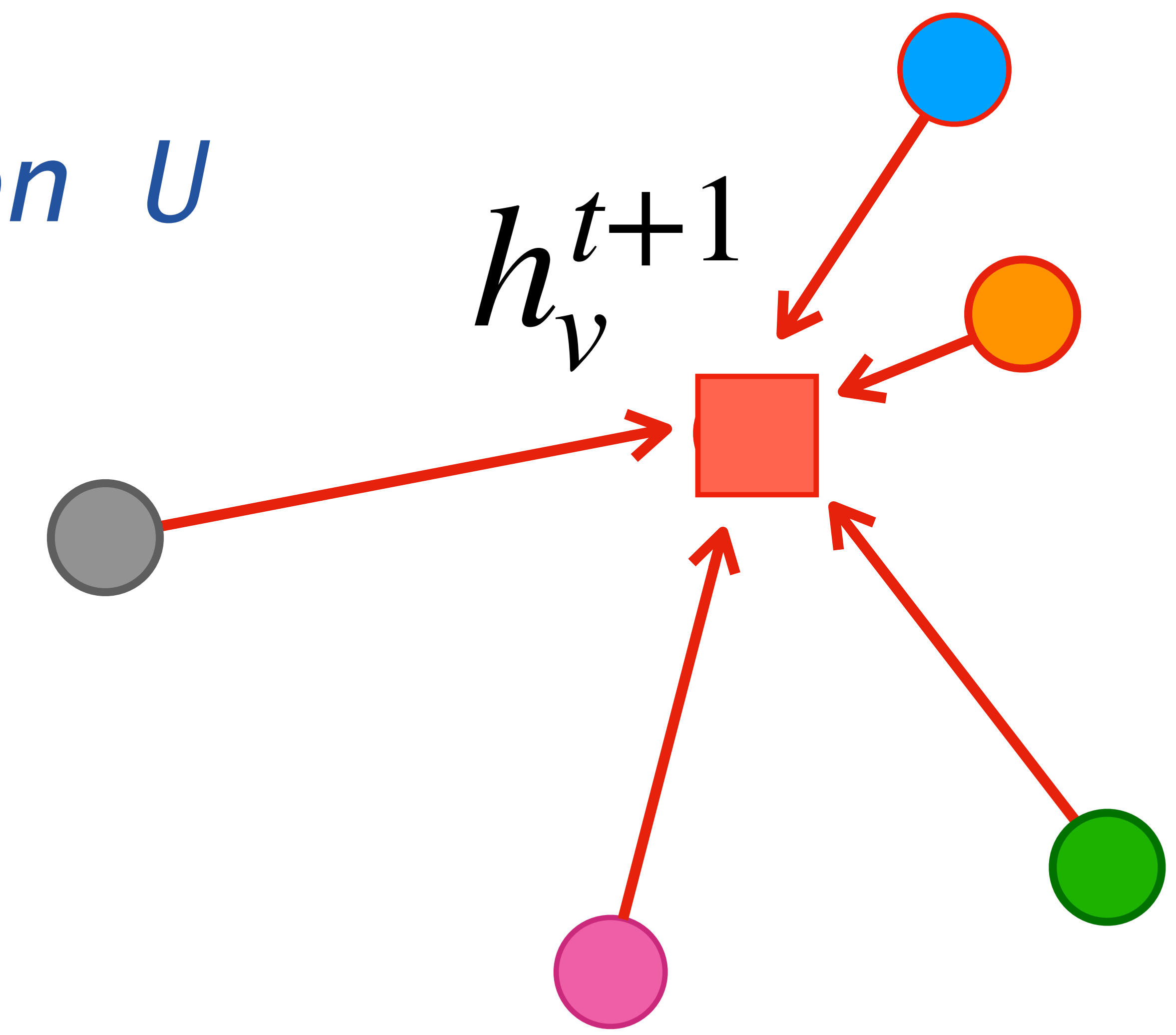


$$m_v^{t+1} = \sum_{w \in G(v)} M_t(h_v^t, h_w^t, e_{vw})$$

With equations...

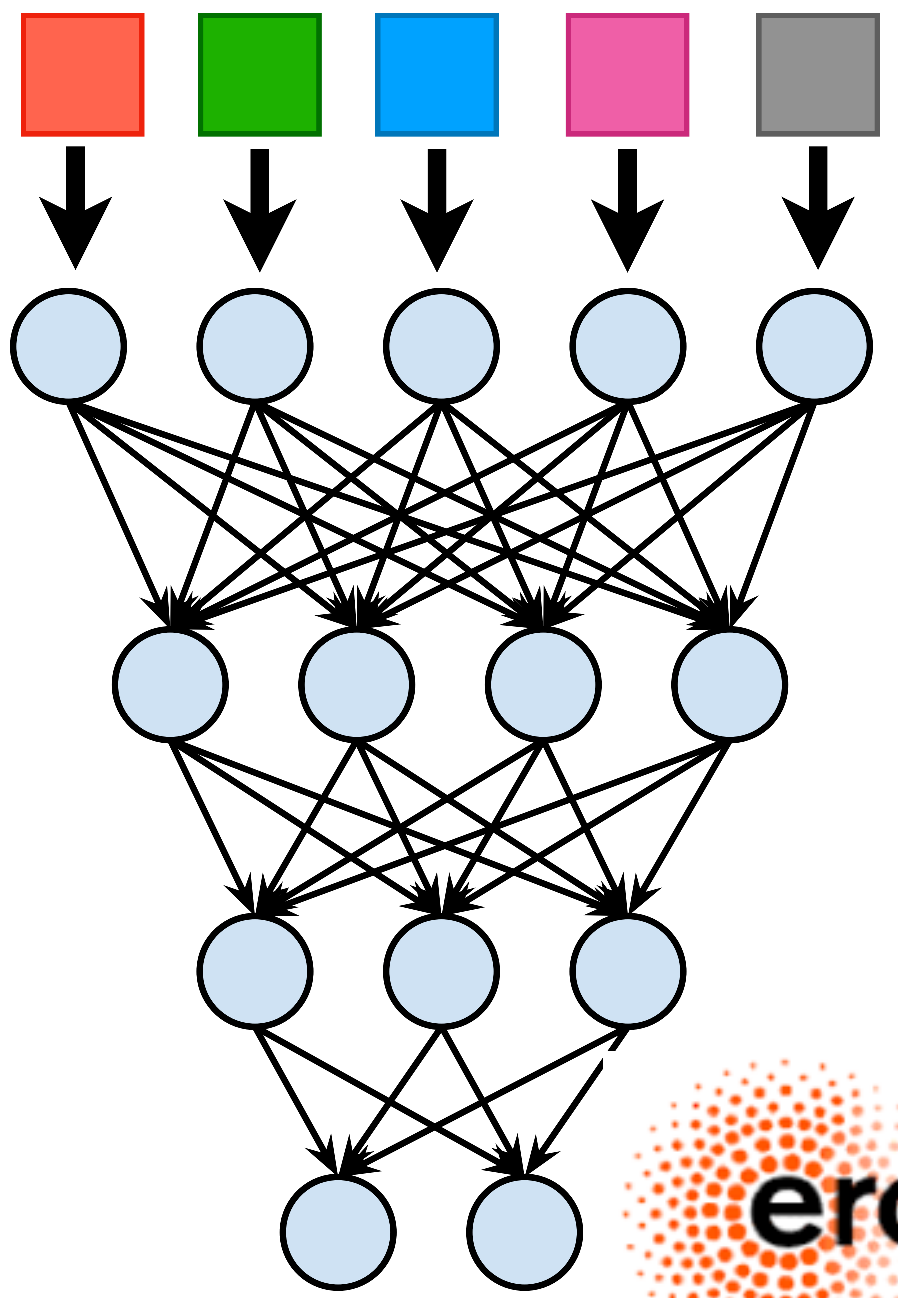
- *The state of vertex v is updated by some function U of the current state and the gathered message*

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$



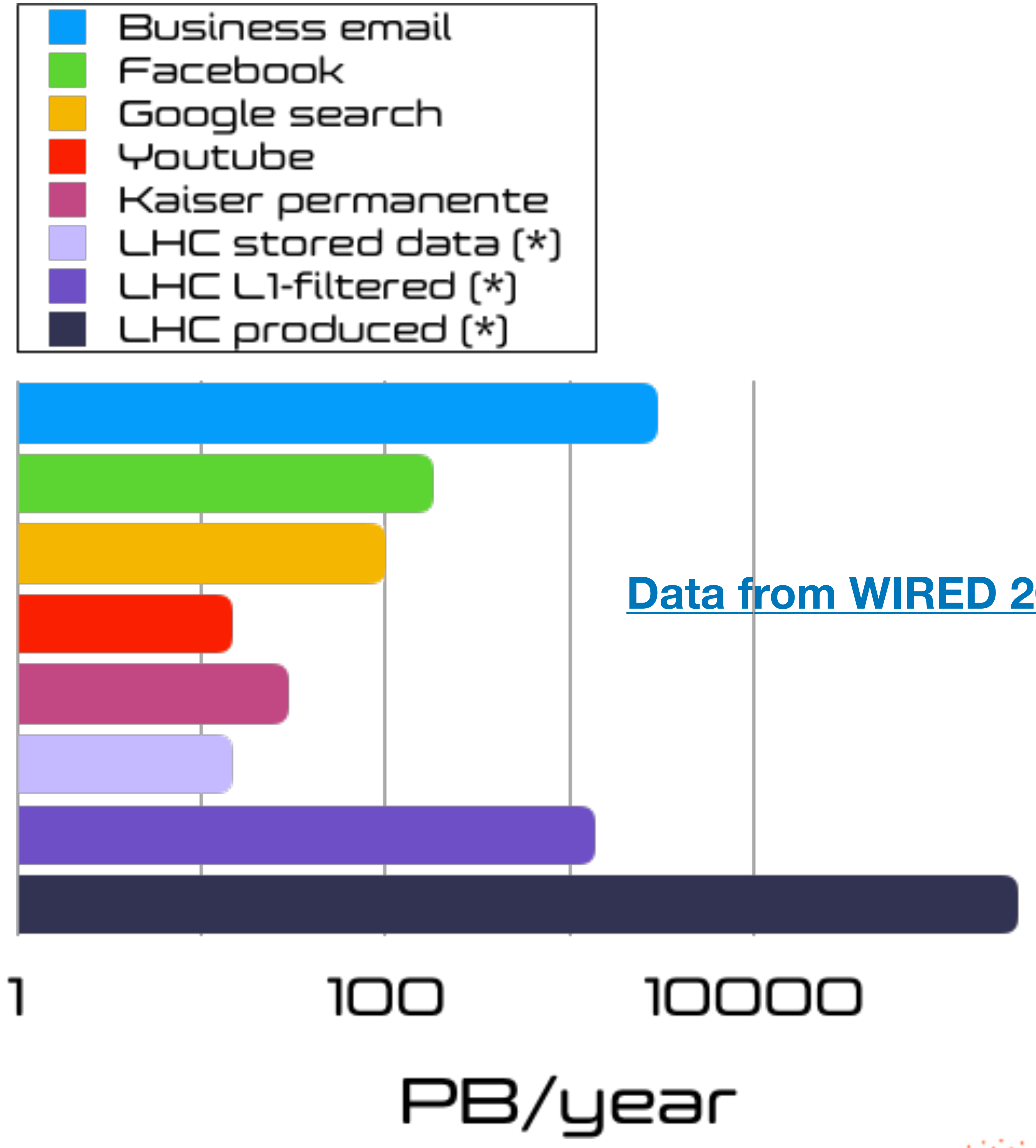
- *After T iterations, the last representations of the graph vertices are used to derive the final output answering the question asked (classification, regression, etc.), typically through a NN*

$$\hat{y} = R(h_v^T \,|\, v \in G)$$



Big Data @LHC

- The amount of produced data is too much to be stored
  - 1,000 times the data generated by google searches+youtube+facebook back in 2013
  - Reduced to 5x(google searches+youtube+facebook) after first filtering
  - Can only store 5% of those

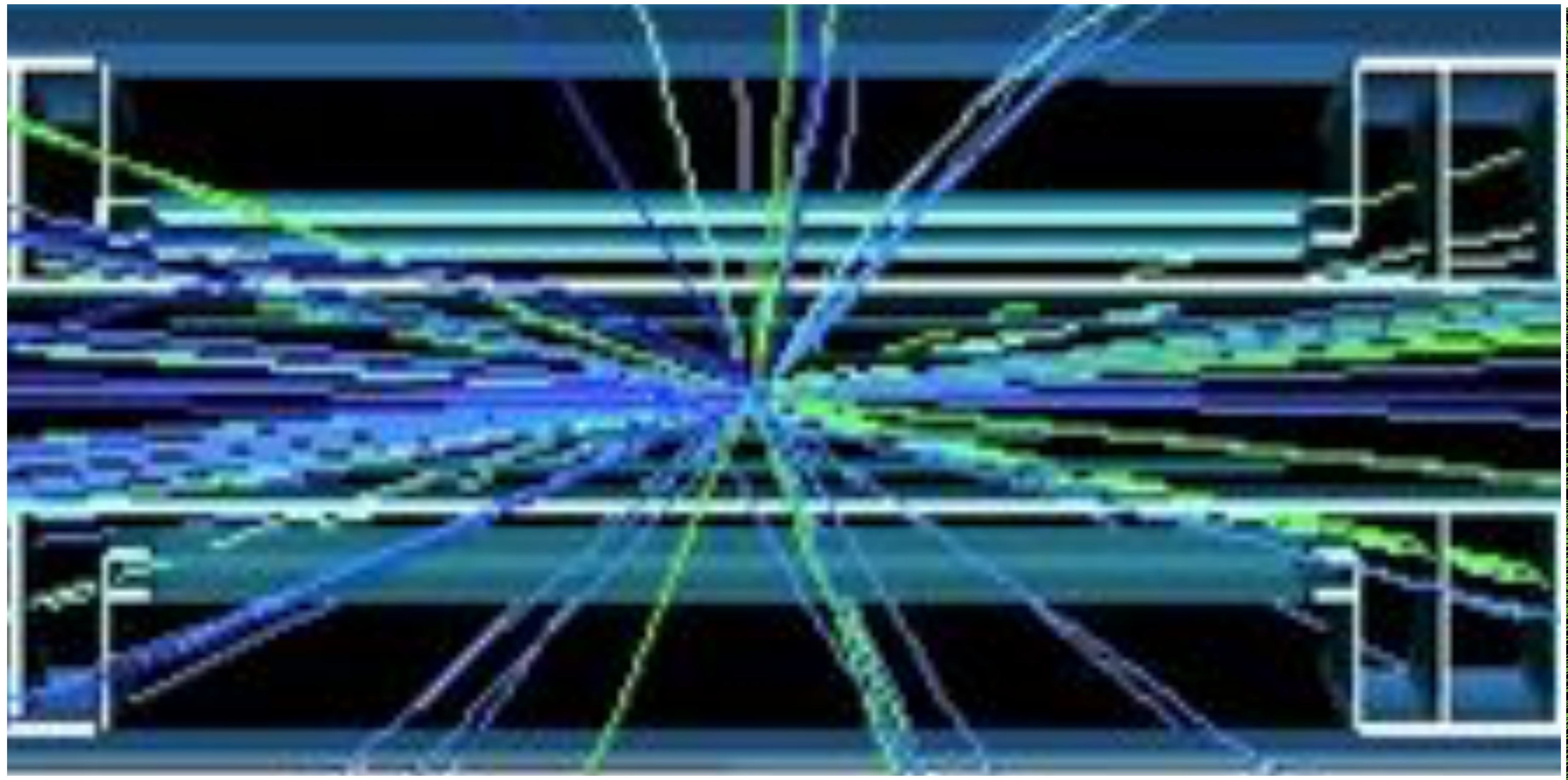


Data from WIRED 2013

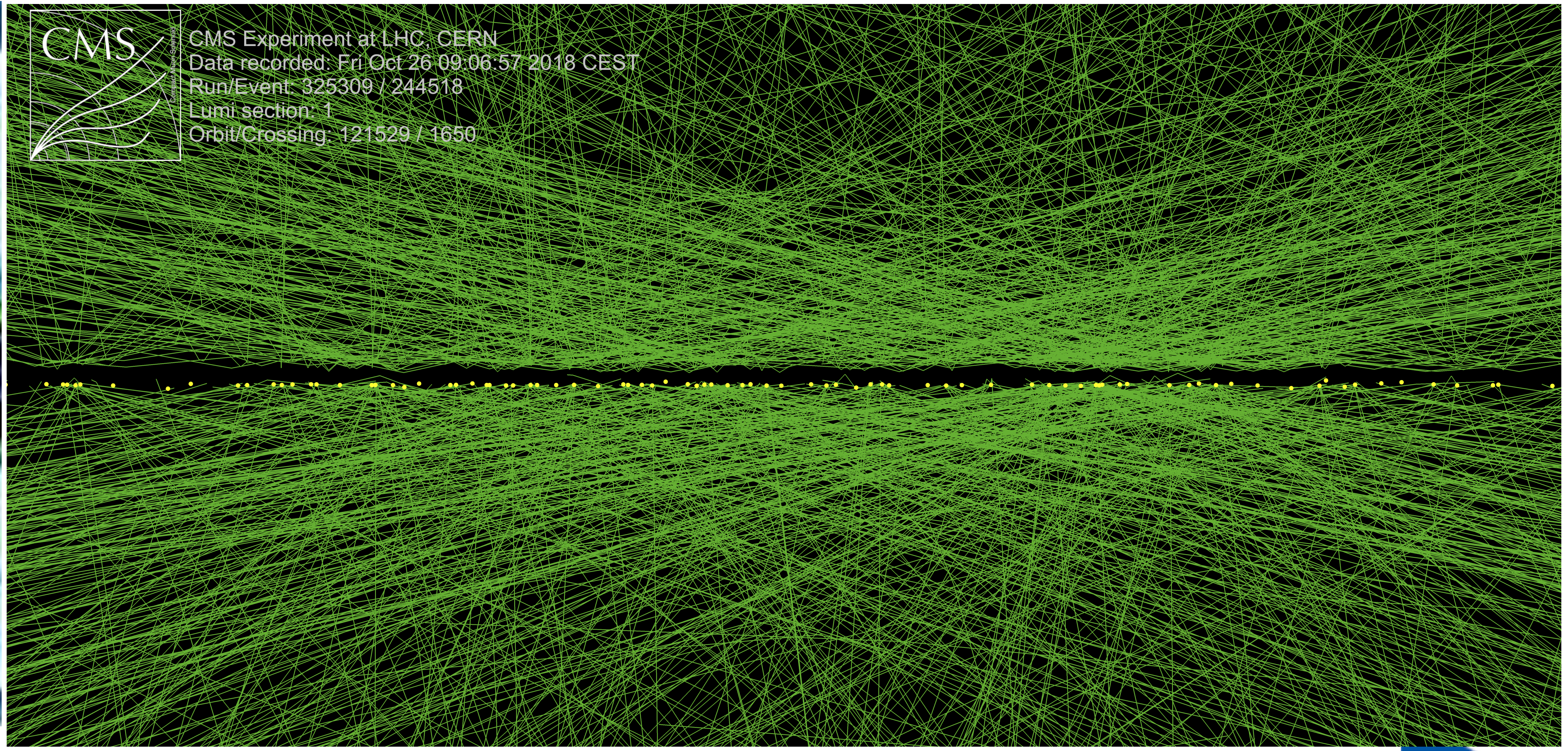
(*) Only two big experiments (ATLAS and CMS), only RAW data

Things will get worse

5 interactions/beam cross



140 interactions/beam cross

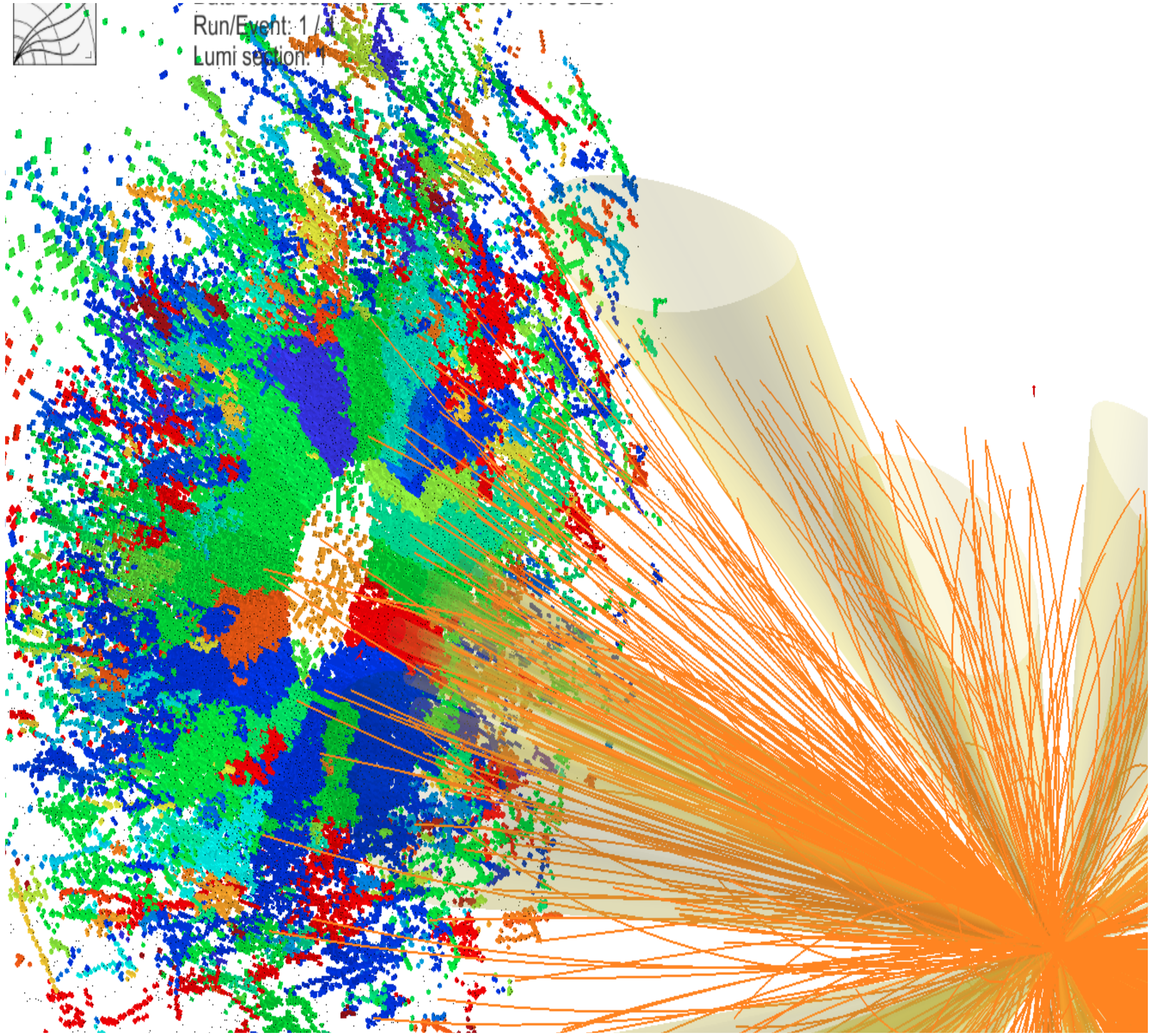


- ~40 collisions/event
- ~10 sec/event processing time
- (at best)Same computing resources as today

- ~200 collisions/event
- ~minute/event processing time
- (at best)Same computing resources as today

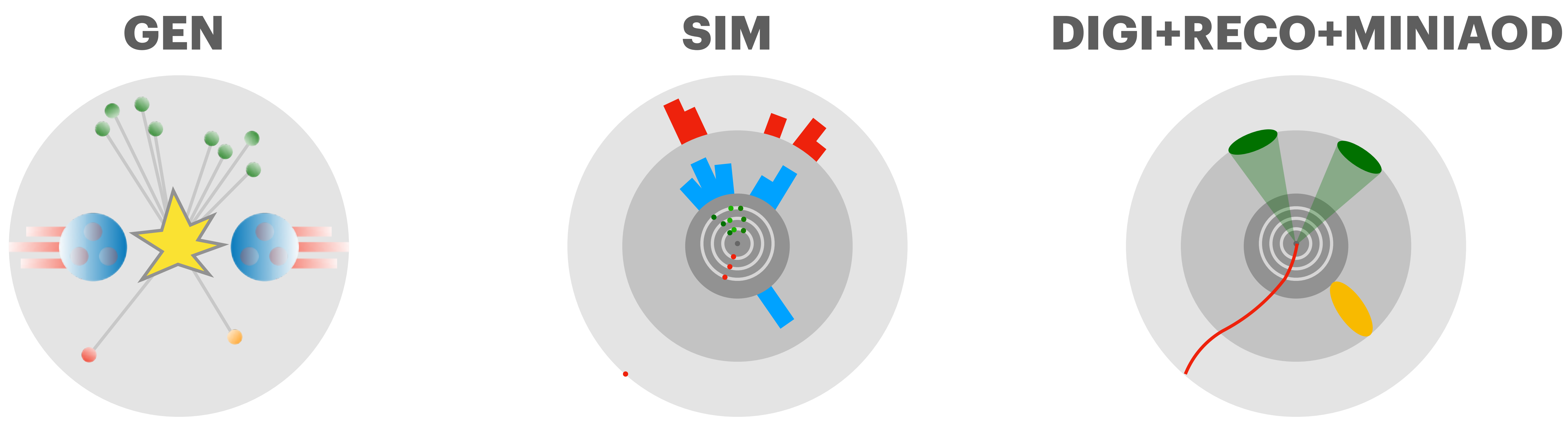
More sensors, more RECO troubles

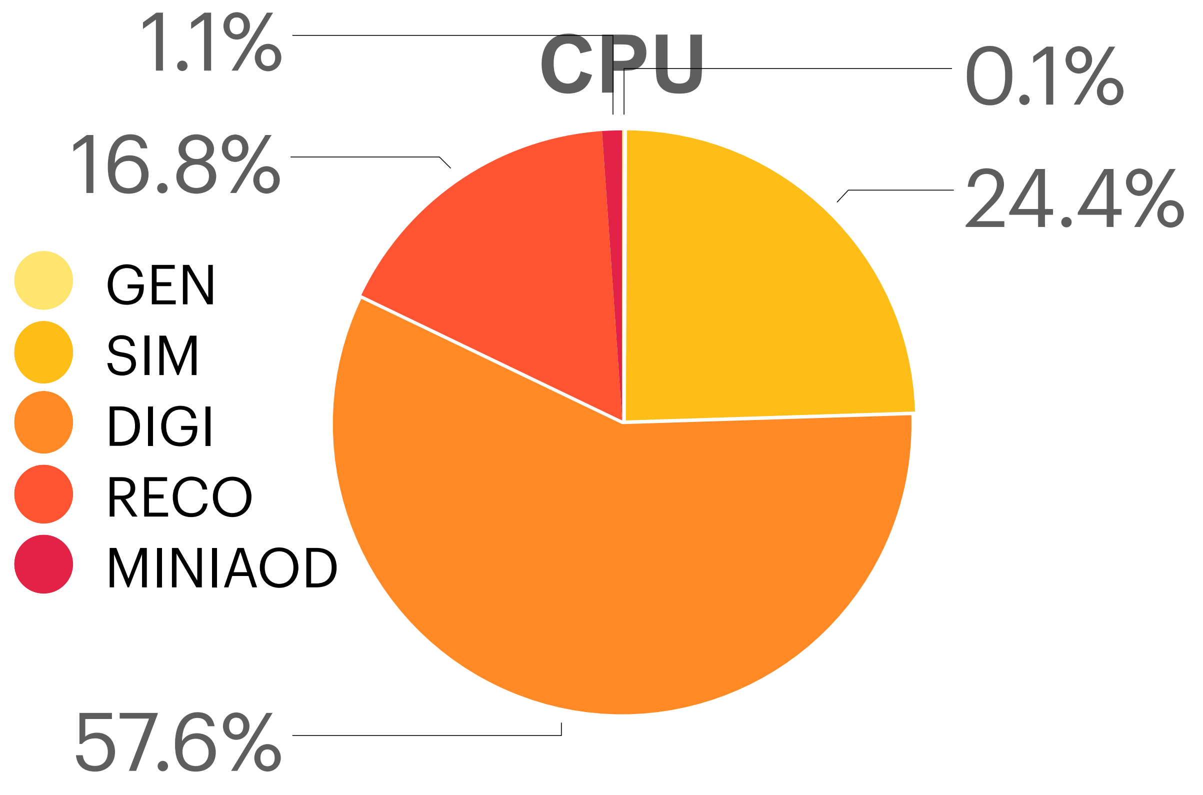
- *To disentangle 200 collisions happening at once, we will build new detectors with more (smaller) sensors*
- *Event complexity grows non linearly*
- *To profit of that, computing resources for data processing will have to increase*
- ***We are off by a factor ~10 if we project to 2027***

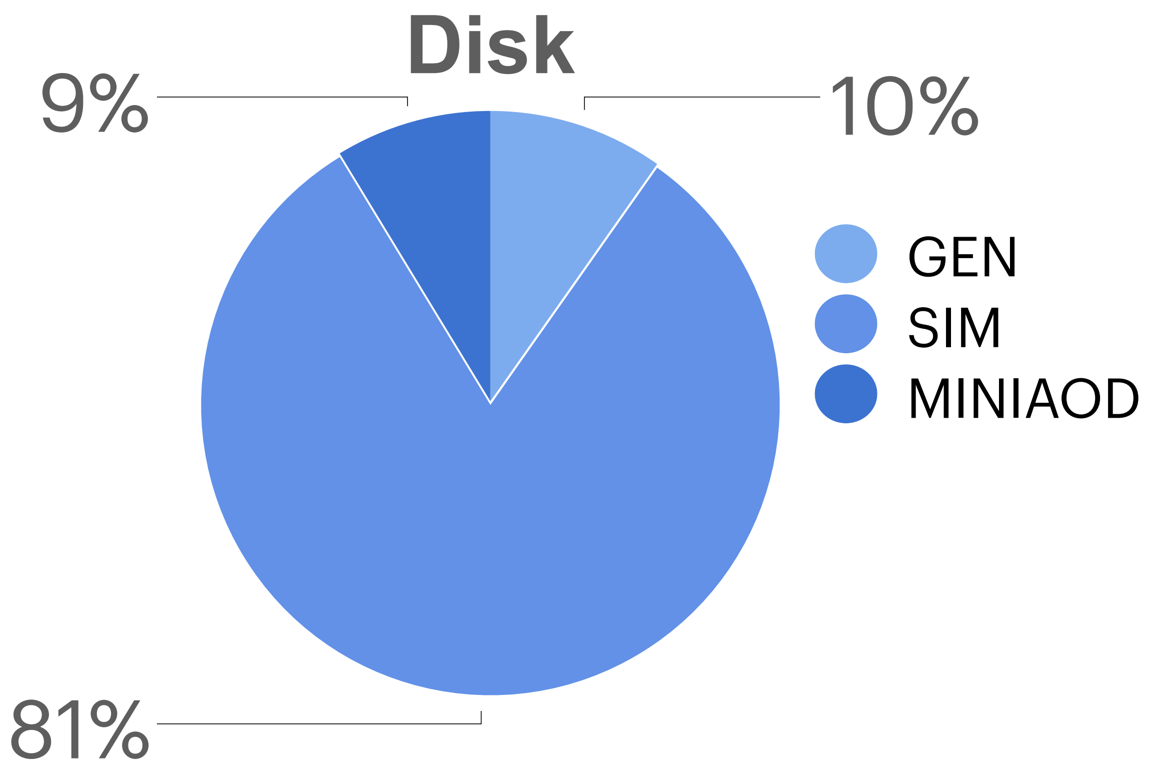


More sensors, more SIM troubles

- *Simulation of LHC collision is essential for analyses*
- *It is a very expensive task, both in terms of CPU & storage*
- *Increasing precision by collecting more data works only if one has more simulation*

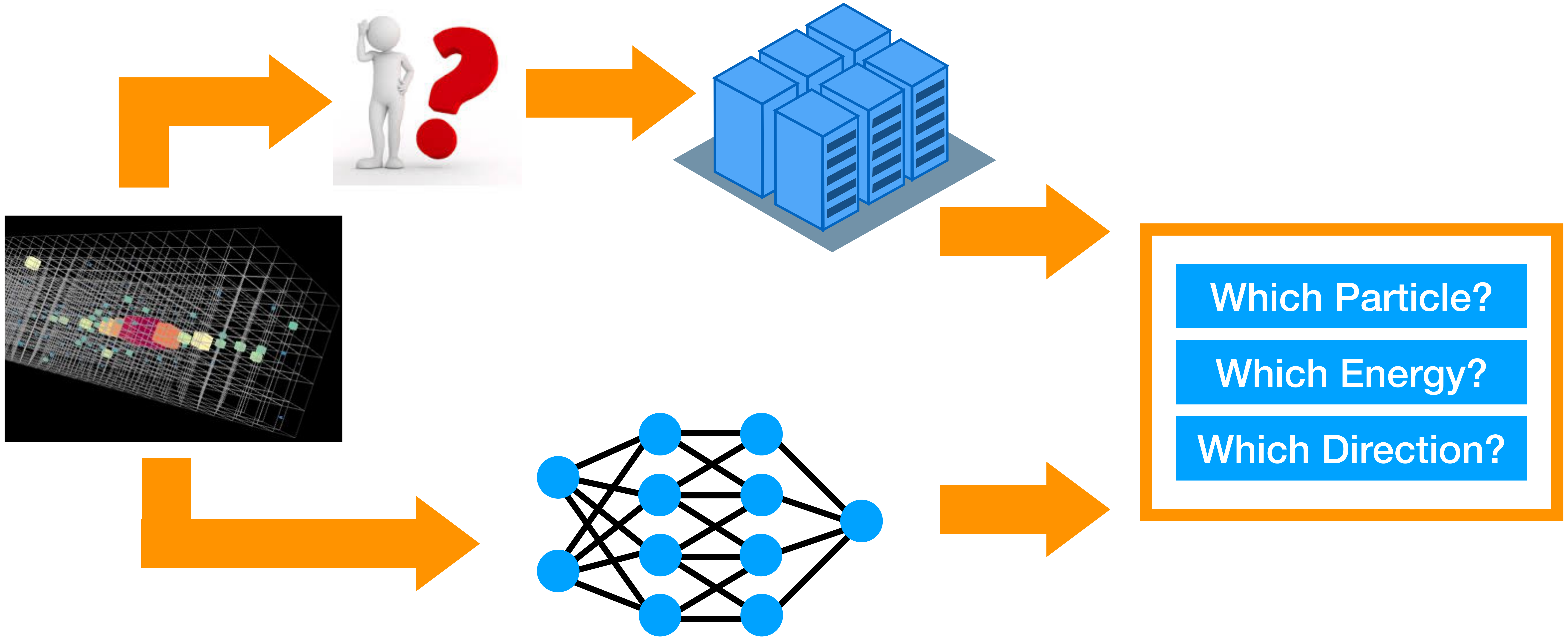






- ***We are off by a factor ~10 if we project to 2027***

Deep Learning at Rescue: Reco



- *We know how to get from the data the answers we want*
  - *physics + intuition + computing*
- *But the process is slow*
  - *We can use DL solutions as a shortcut: we teach neural networks how to give us the answer we want directly from the raw data*

Deep Learning at Rescue: Sim

- *We know how to get from the data the answers we want*
  - *physics + intuition + computing*
- *But the process is slow*
  - *We can use DL solutions as a shortcut: we teach neural networks how to give us the answer we want directly from the raw data*