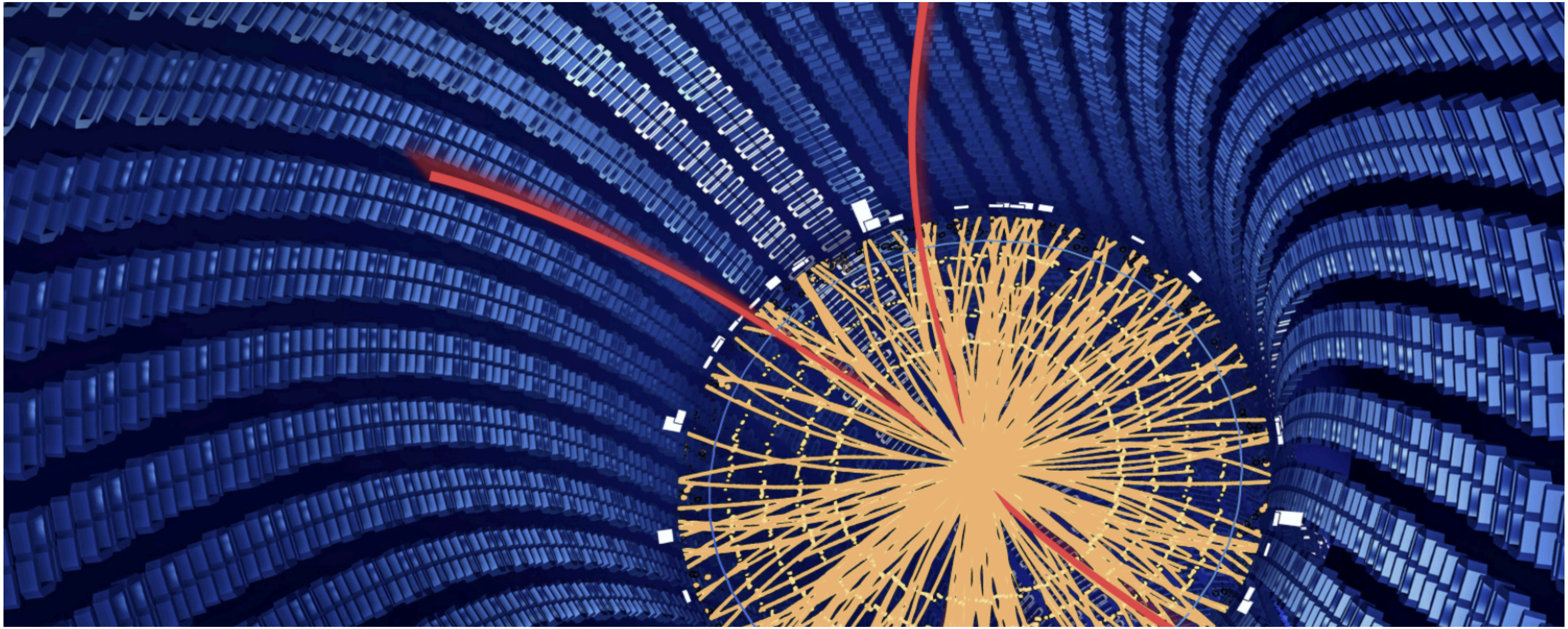
Deep Learning Applications for collider physics
Lecture 4

Maurizio Pierini

European Research Council

Plan for this week

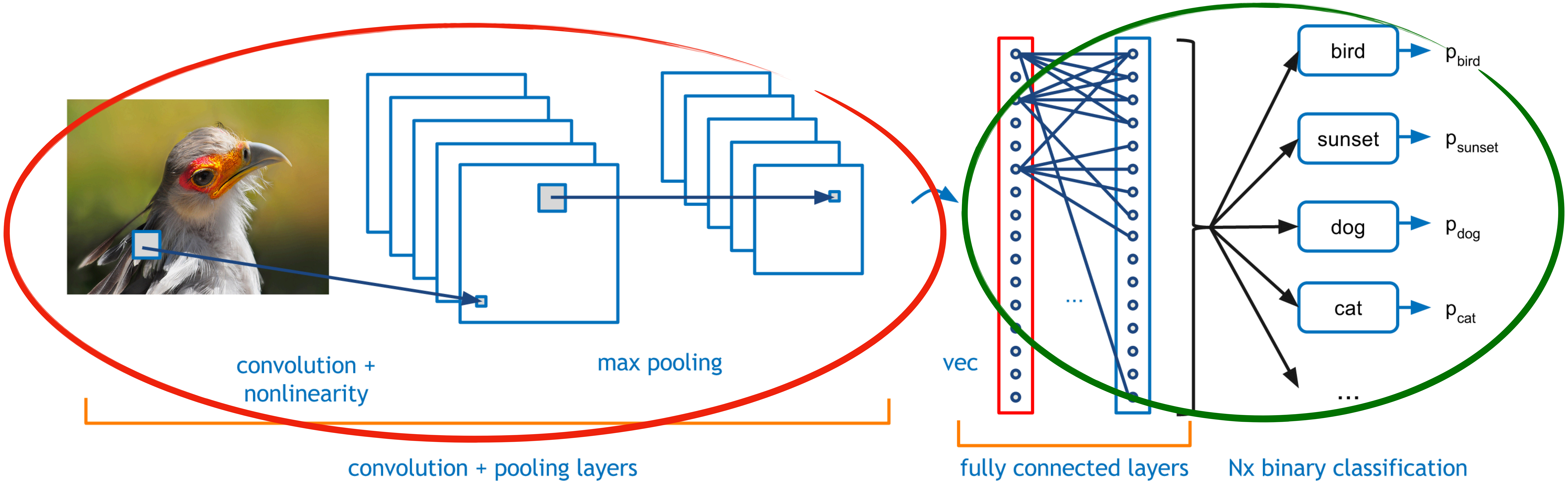
	Day1	Day2	Day3	Day4	Day5
Lecture	Introduction	ConvNN	RNNs	Graphs	Unsupervised Learning
Tutorial	Fully Connected Classifier	ConvNN Classifier	RNNs Classifier	Graphs Classifier	Anomaly Detection



HEP data and Graphs

Deep Neural Network in a nutshell

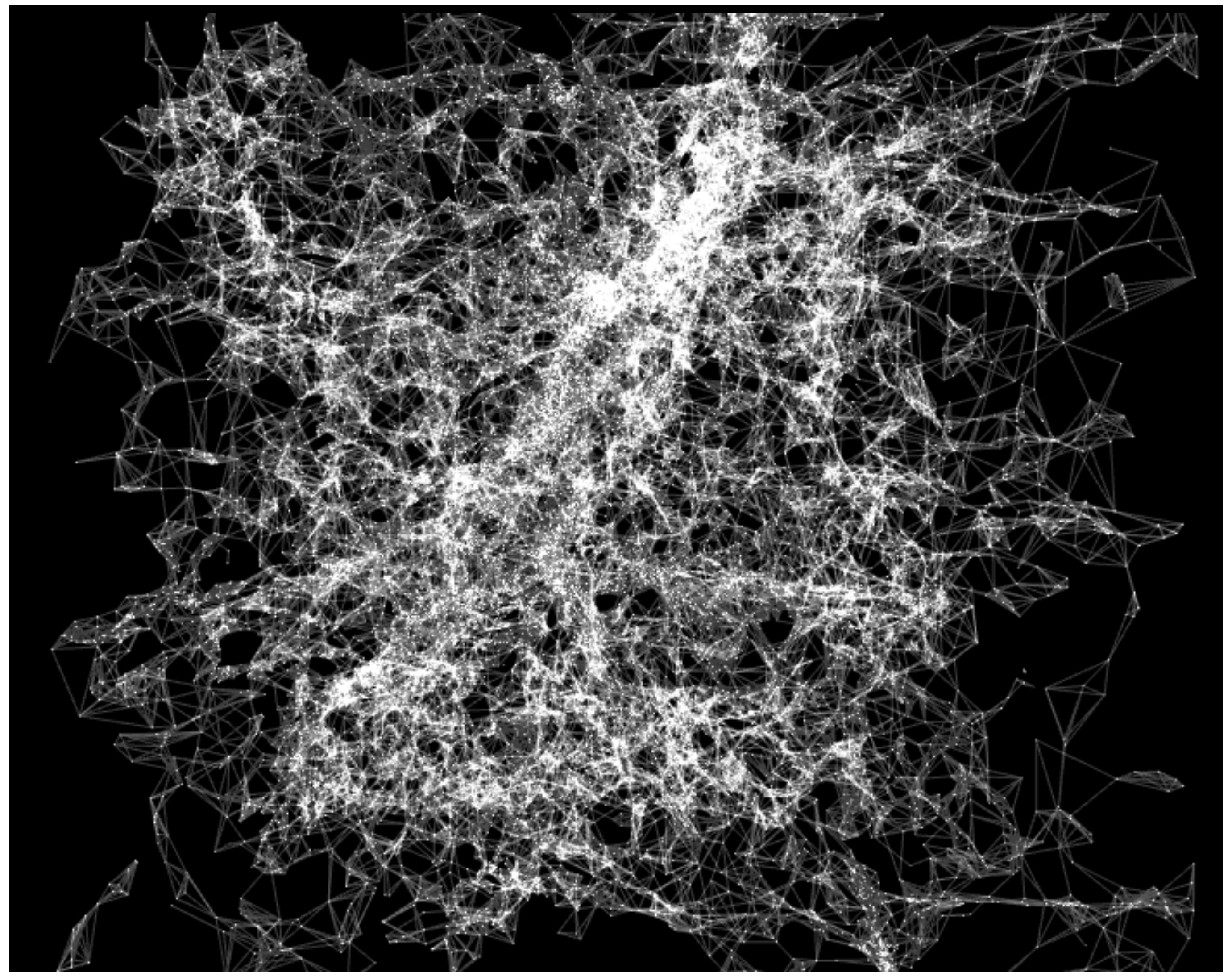
- DNNs typically rely on two phases:
  - ***Feature engineering*** *from Raw Data. This is where new & exotic architectures (depending on data type) take the best out of your data*

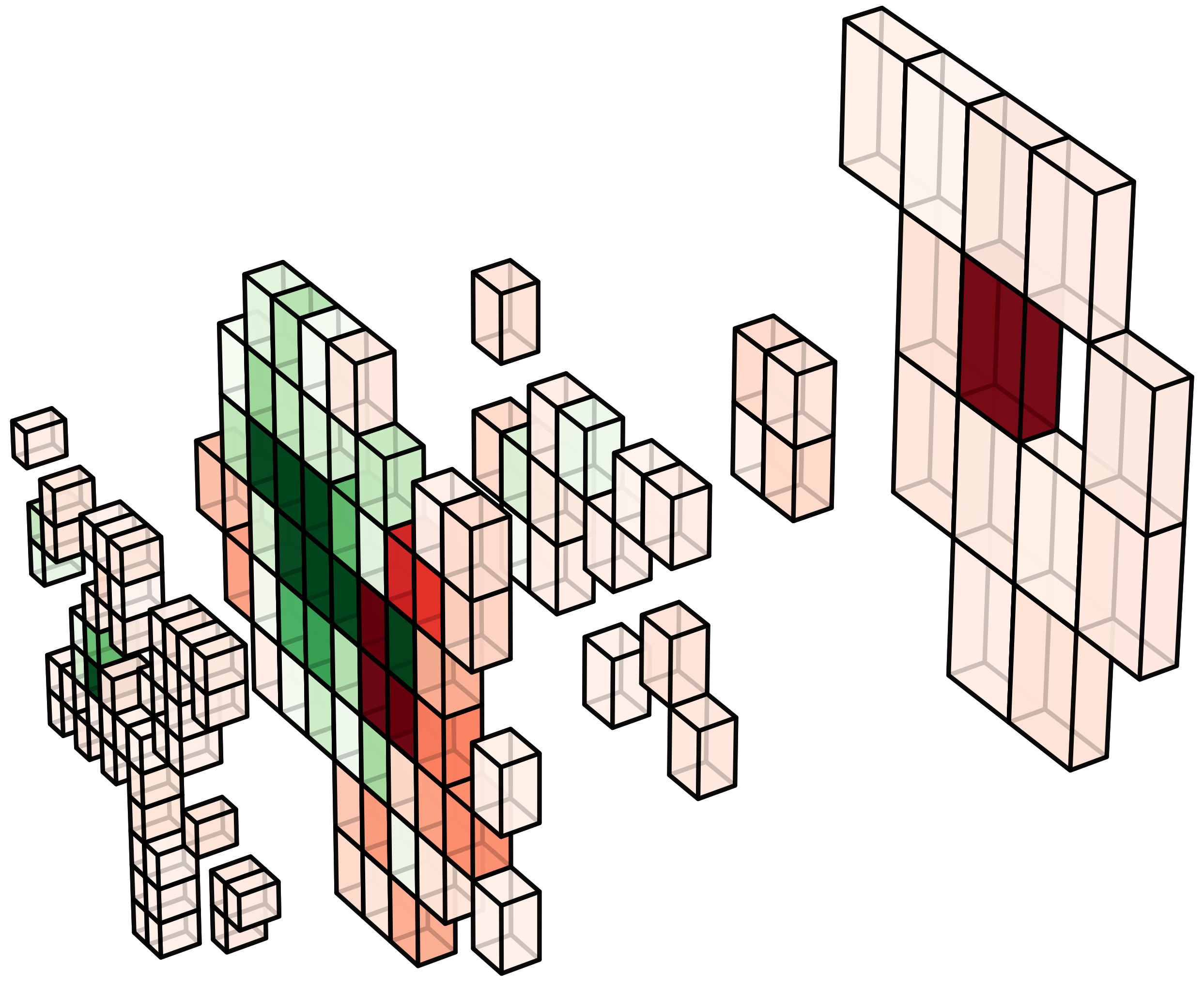


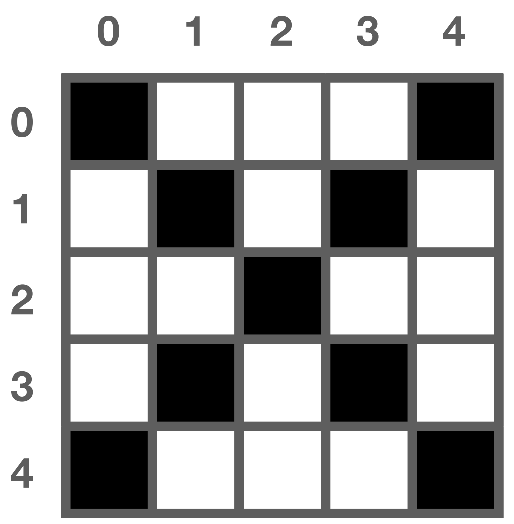
  - ***Task solving****: start from engineered features and solve the task (classification, regression, etc.)*

What about irregular data?

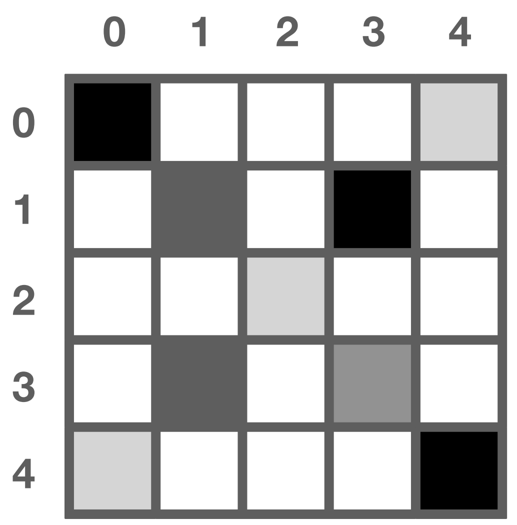
- Unfortunately, many scientific domains deal with data which are not regular arrays (neither images nor sequences)
  - Galaxies or star populations in sky
  - Sensors from HEP detector
  - Molecules in chemistry
- These data can all be seen as sparse sets in some abstract space
  - each element of the set being specified by some array of features
  - geometrical coordinates could be some of these features







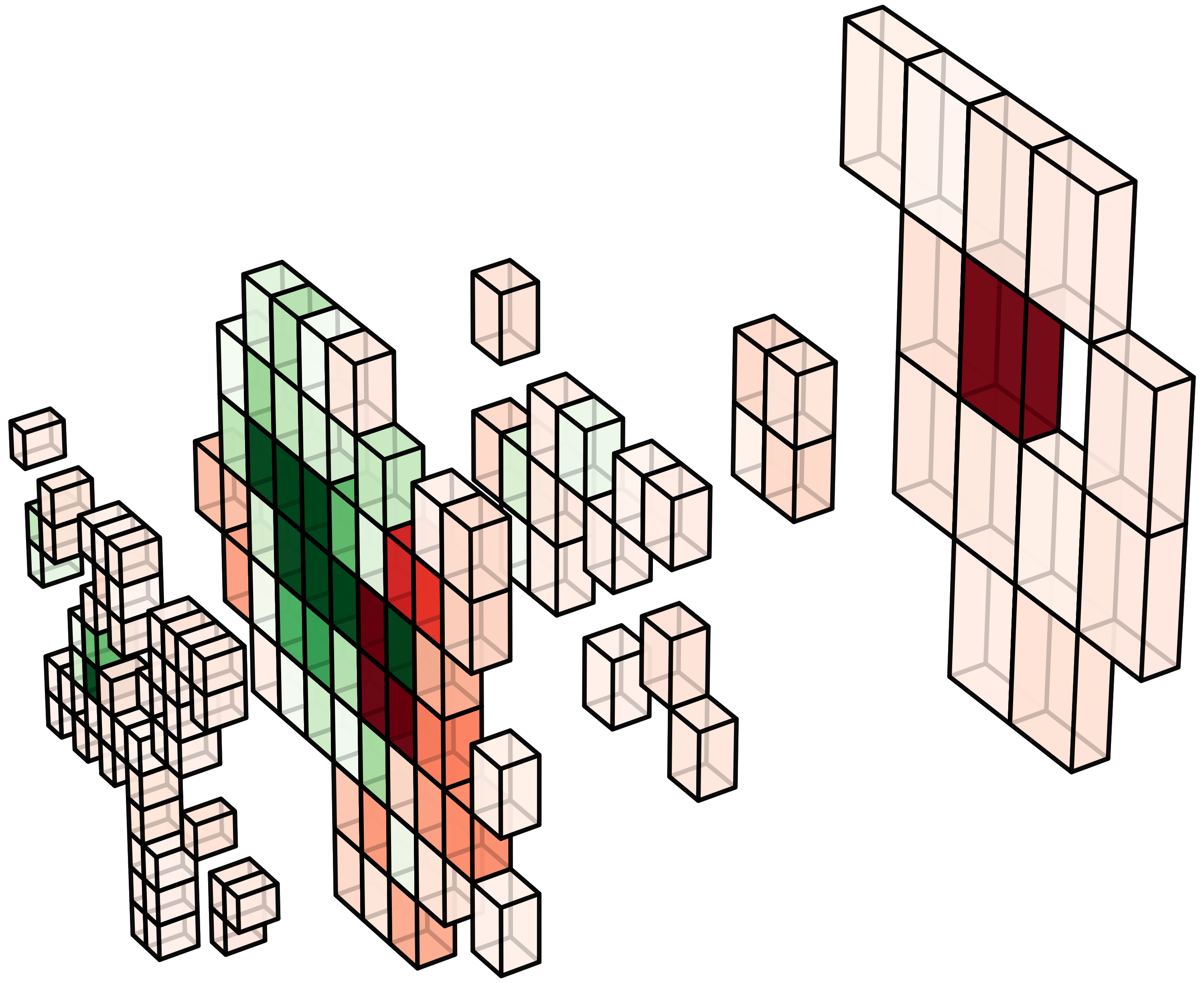
0	0	1	1	2	3	3	4	4
0	4	1	3	2	1	3	0	4



0	0	1	1	2	3	3	4	4
0	4	1	3	2	1	3	0	4
1.00	0.25	0.75	1.00	0.25	0.75	0.50	0.25	1.00

European Research Council

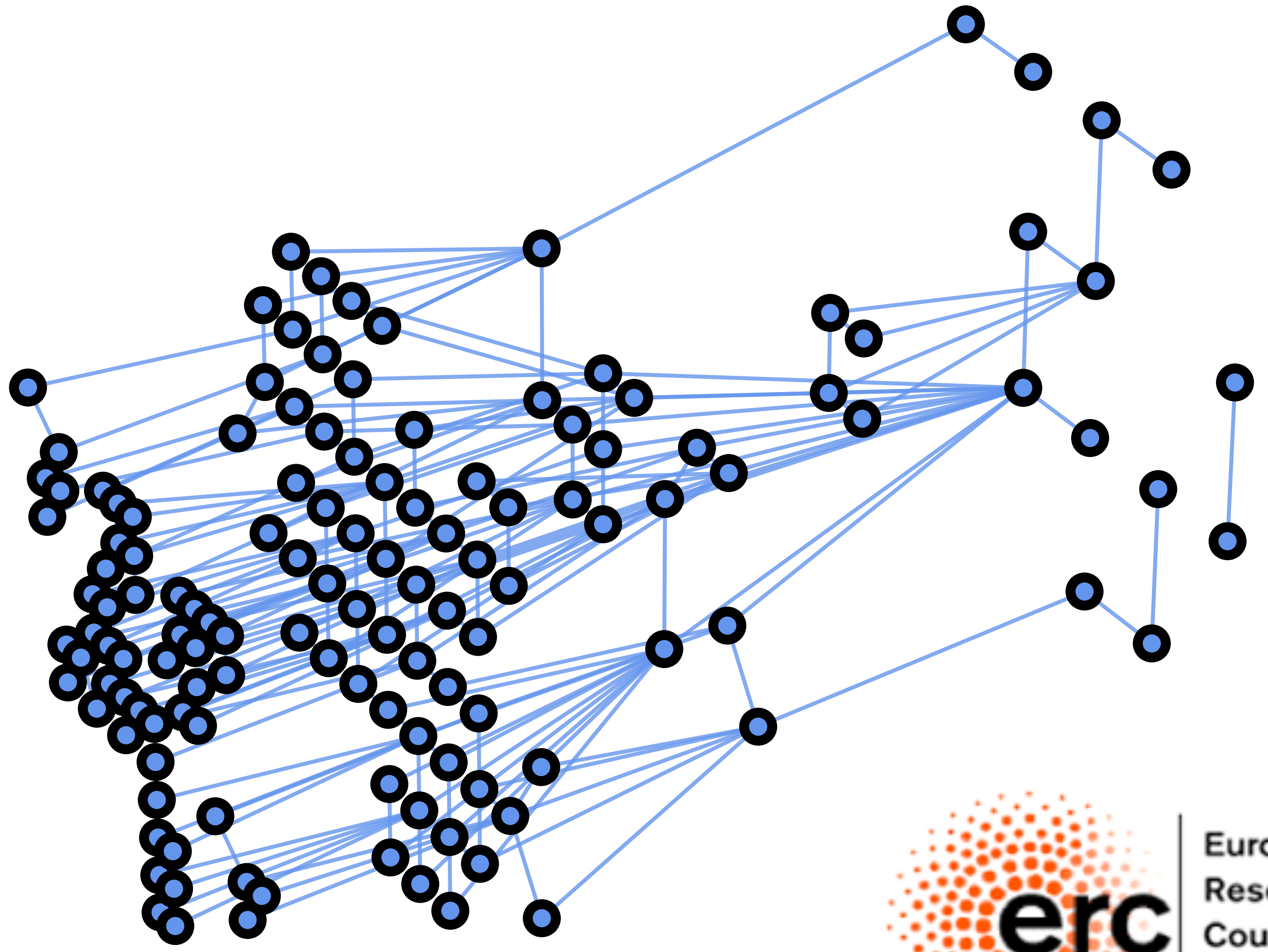
From Sets to Graphs



- *Given such a set, we want to generalise the image representation as regular array that is fed to a CNN*

    - *Once that is done, we can generalise CNN itself*

- *For images, a lot of information is carried by pixels being next to each other. A metric is intrinsic in the data representation as image*



- *With a set, we need to specify a metric that tell us who is close to who in the abstract space of features that we have at hand*

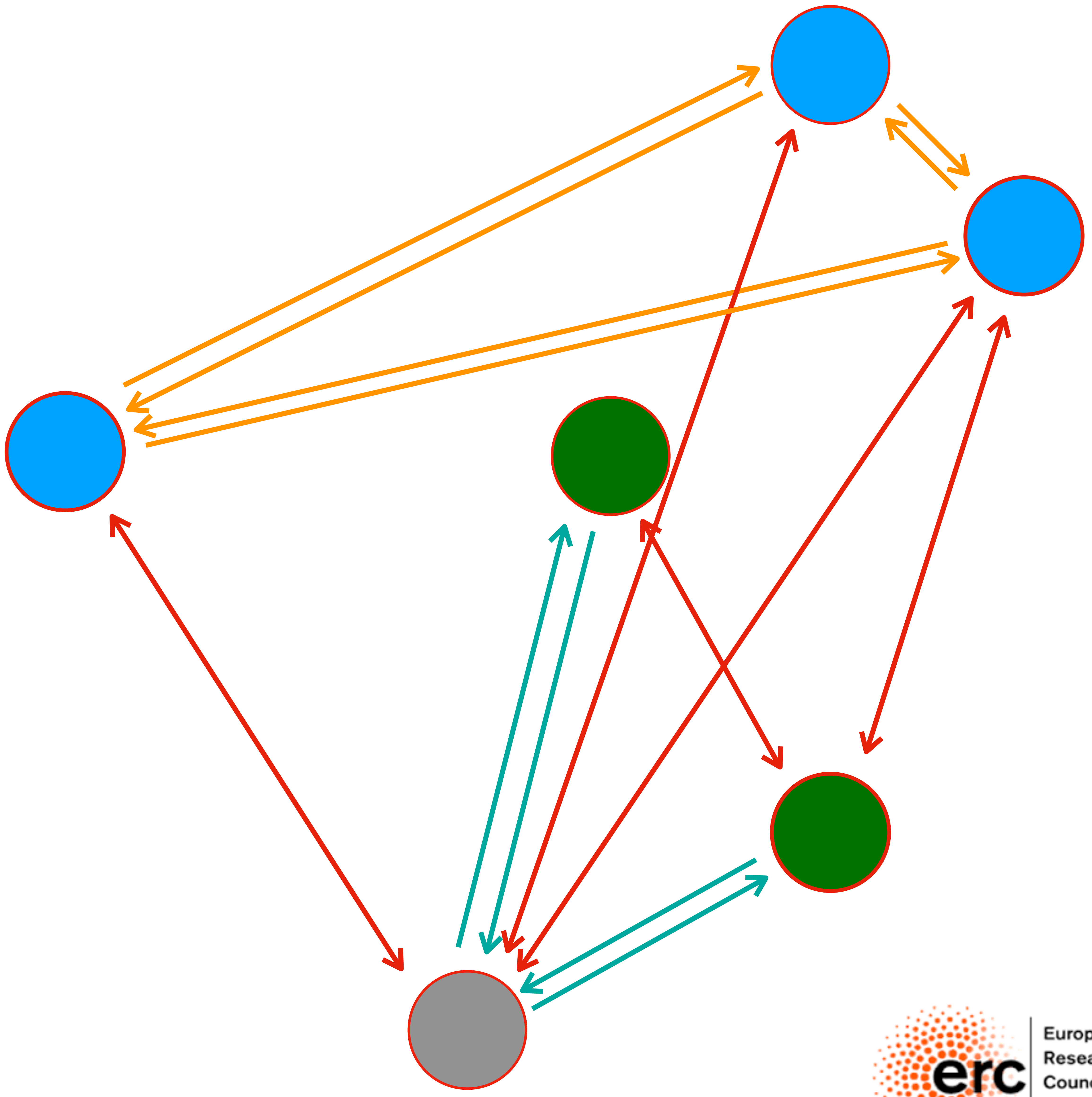
    - *SOLUTION: connect elements of sets and learn (e.g., with a neural network) from data which connections are relevant*

CERN

European Research Council

Building the Graph

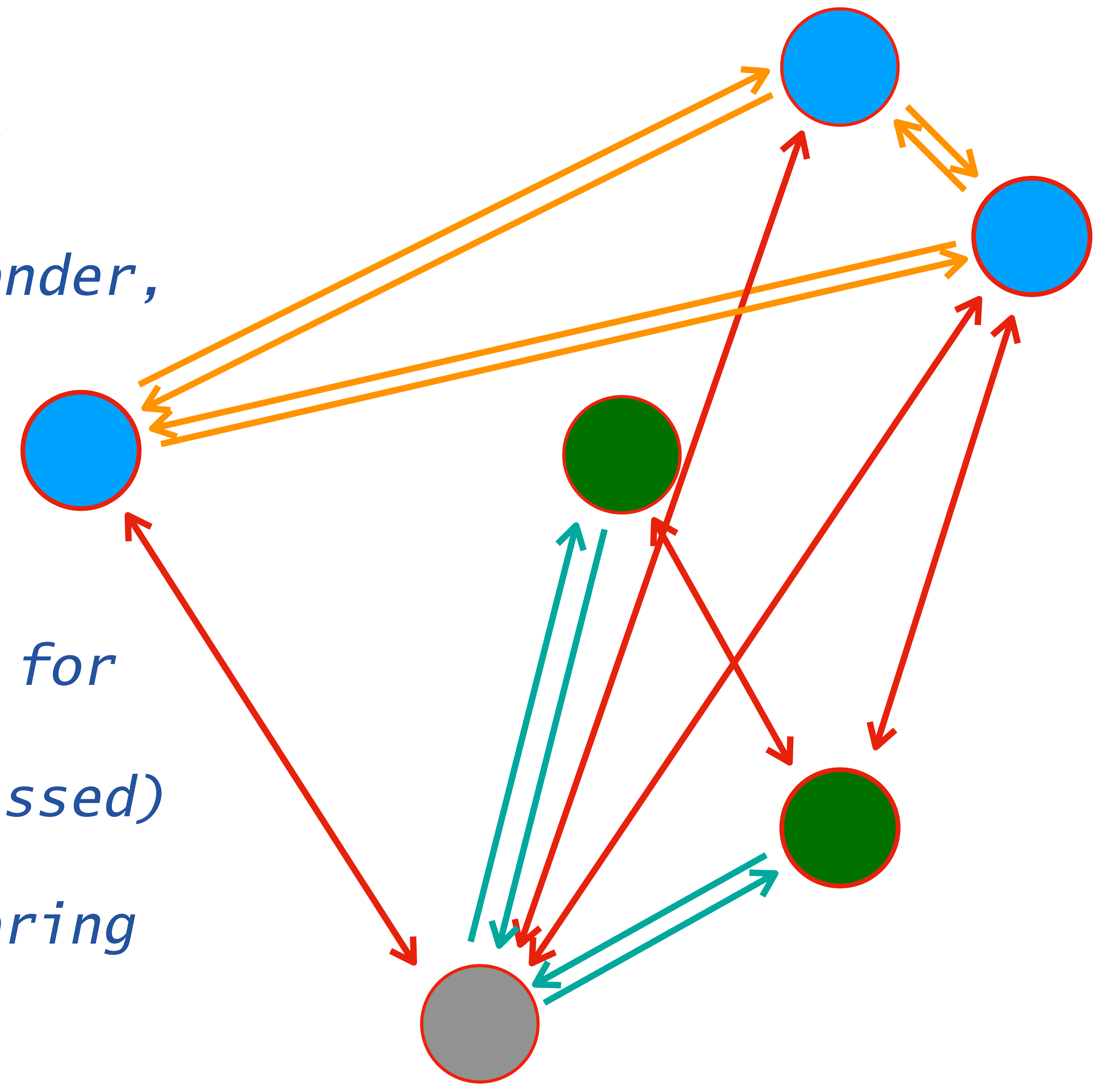
- Each element of your set is a vertex V
- Edges E connect them
  - Edges can be made directional
  - Graphs can be fully connected (N^2)
  - Or you could use some criterion (e.g., nearest k neighbours in some space) to reduce number of connections
  - if more than one kind of vertex, you could connect 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



Message Passing

Learning from Graph: an example

- *Imagine a concrete example: given a social-media user, who will she vote for at the next elections?*
- *The graph here comes from social-media connections*
- *The features are what we know for a given user (gender, age, education, etc.)*
- *We want to gather information on someone from the social network of that person*
    - *we might know who some of her connections voted for*
- *We will use NNs to model the influence (message passed) of each user on her connection and learn from data which are the relevant connections. We are engineering features*
- *A final classifier will give us the answer we want*
- *You might become president with this + target pressure (ads, fake news, etc.)*

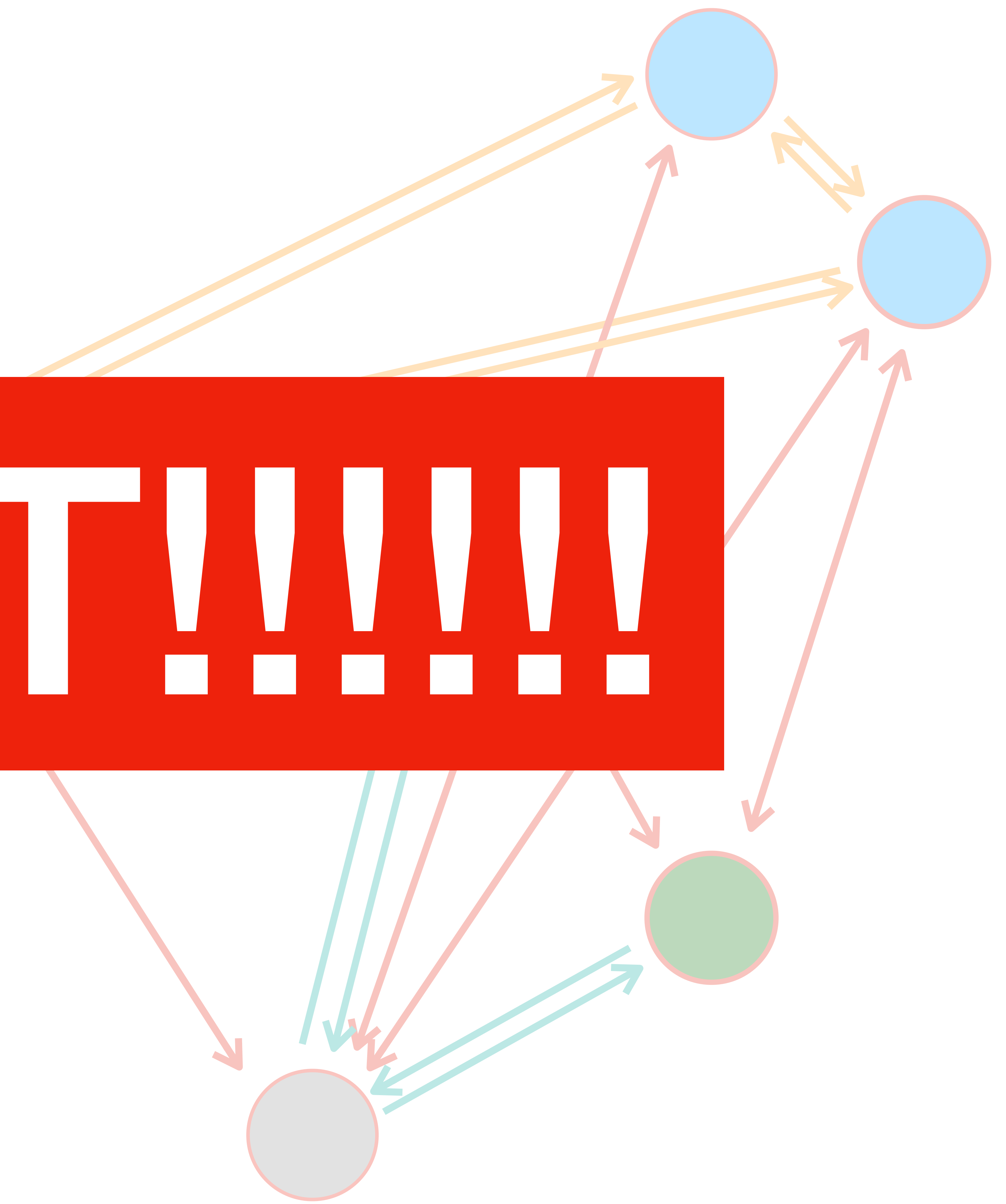


Learning from Graph: an example

- Imagine a concrete example: given a social-media user, who will she vote for at the next elections?
- The graph here comes from social-media connections
- The features are what we know for a given user (gender, age, education, etc.)
- We w…
  soci…
  - we…

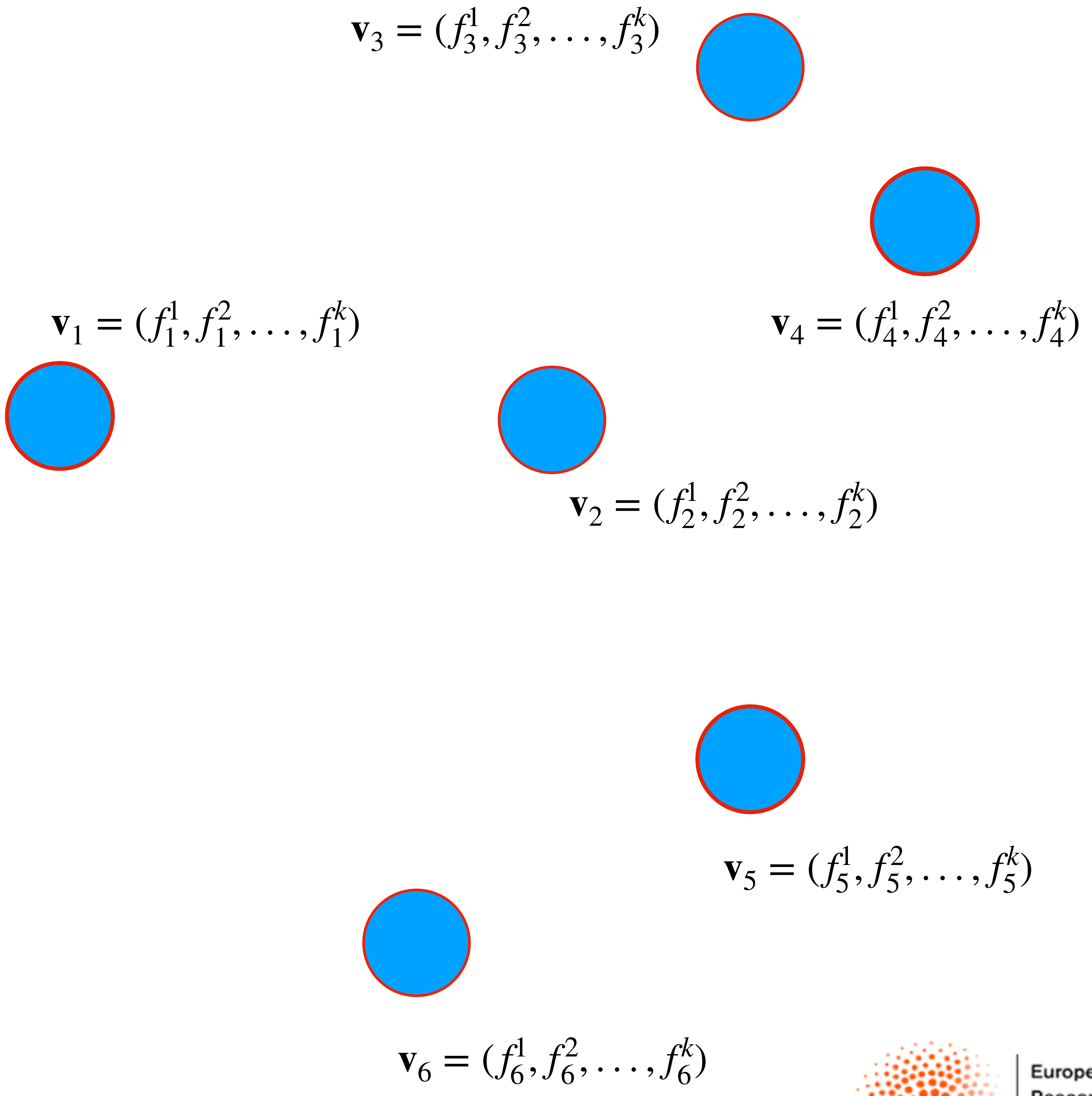
DON'T DO IT!!!!!!

- We will use NNs to model the influence (message passed) of each user on her connection and learn from data which are the relevant connections. We are engineering features
- A final classifier will give us the answer we want
- You might become president with this + target pressure (ads, fake news, etc.)



Graph Networks

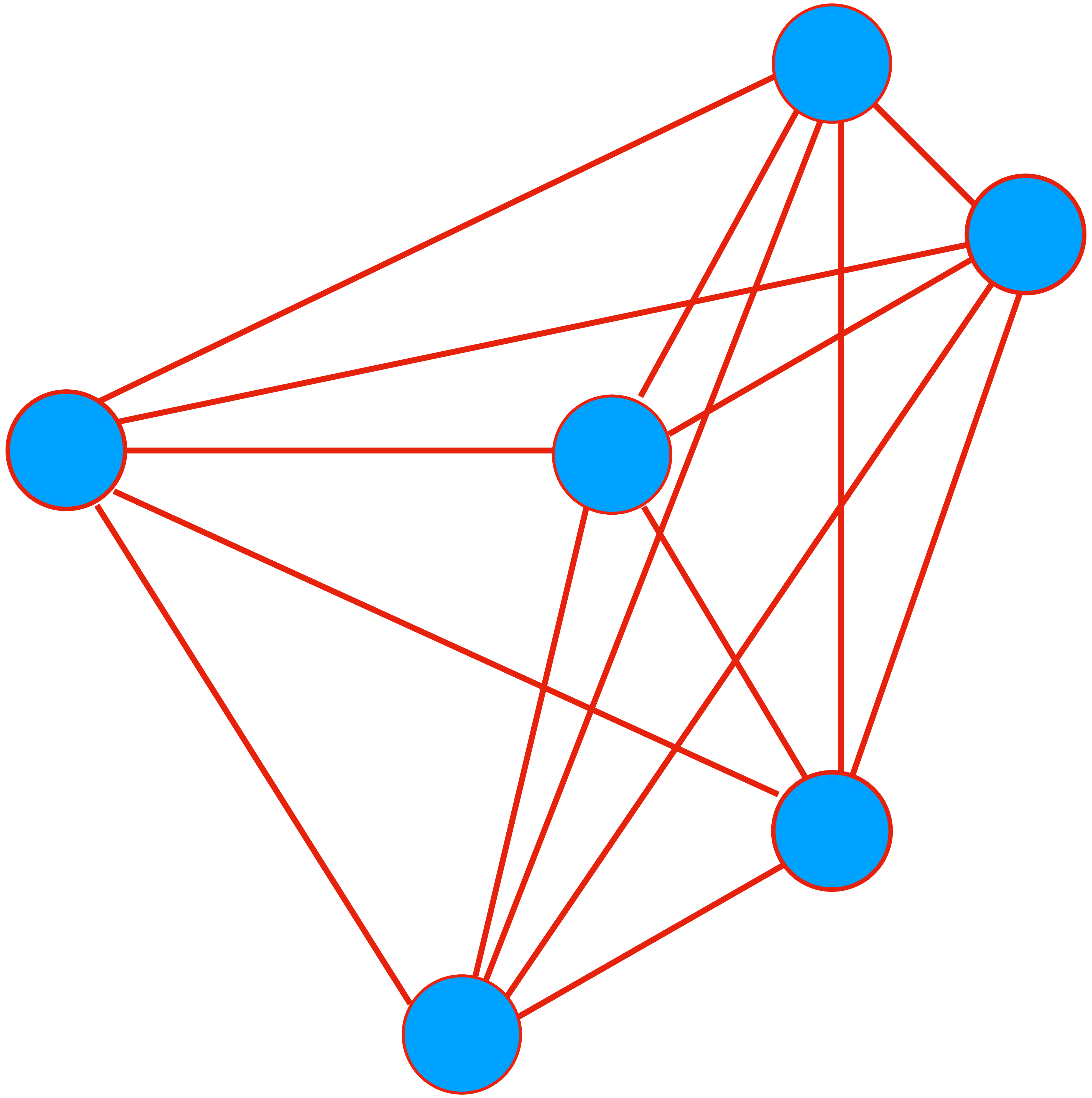
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - ***Each example in a dataset is represented as a set of vertices***
  - ***Each vertex is embedded in the graph as a vector of features***



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

Graph Networks

- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - *Each example in a dataset is represented as a set of vertices*
  - *Each vertex is embedded in the graph as a vector of features*
  - ***Vertices are connected through links (edges)***



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

Graph Networks

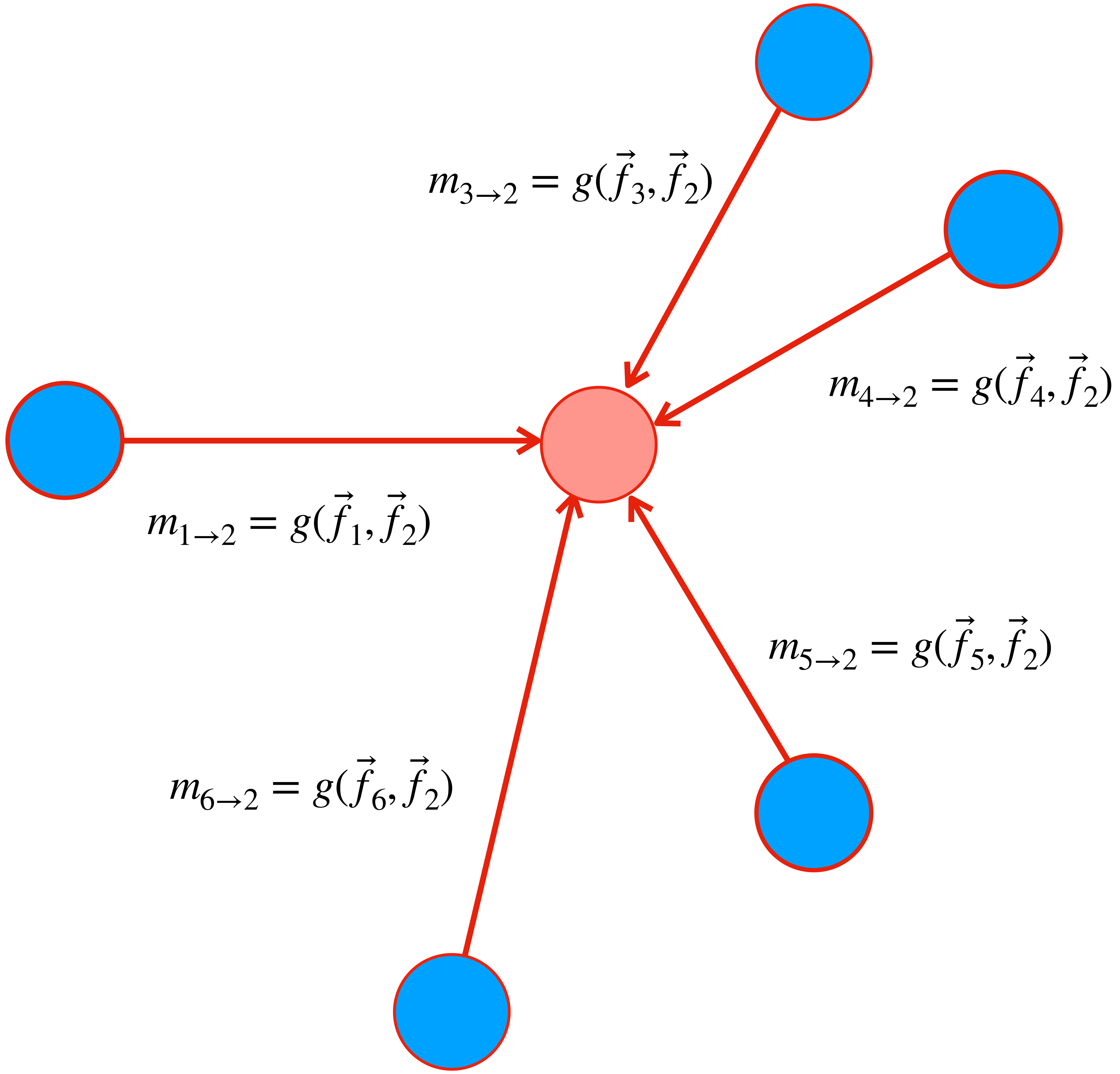
- Graphs Nets are architectures based on an abstract representation of a given dataset

  - Each example in a dataset is represented as a set of vertices

  - Each vertex is embedded in the graph as a vector of features

  - Vertices are connected through links (edges)

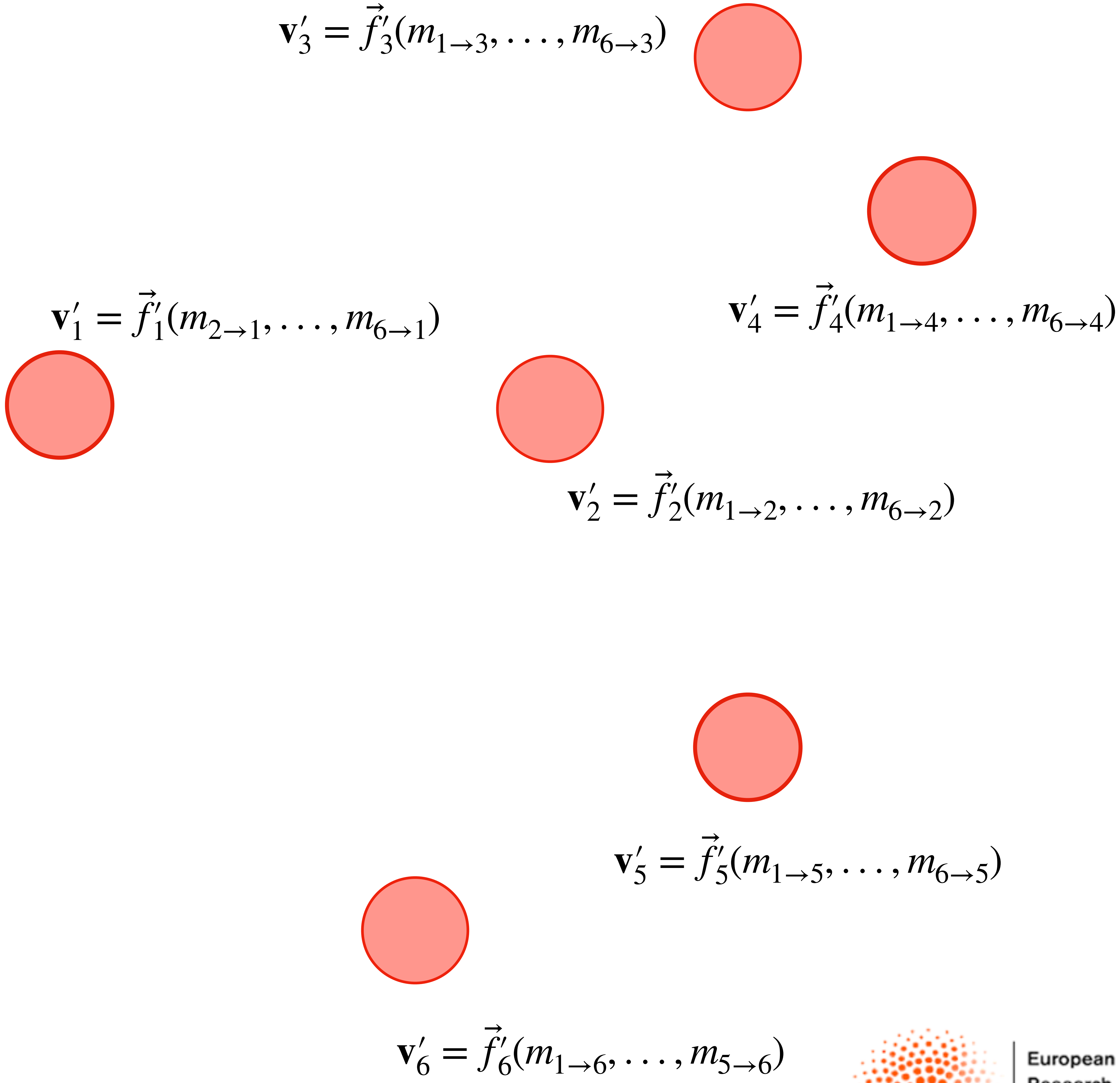
  - **Messages are passed through links and aggregated on the vertices**



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

Graph Networks

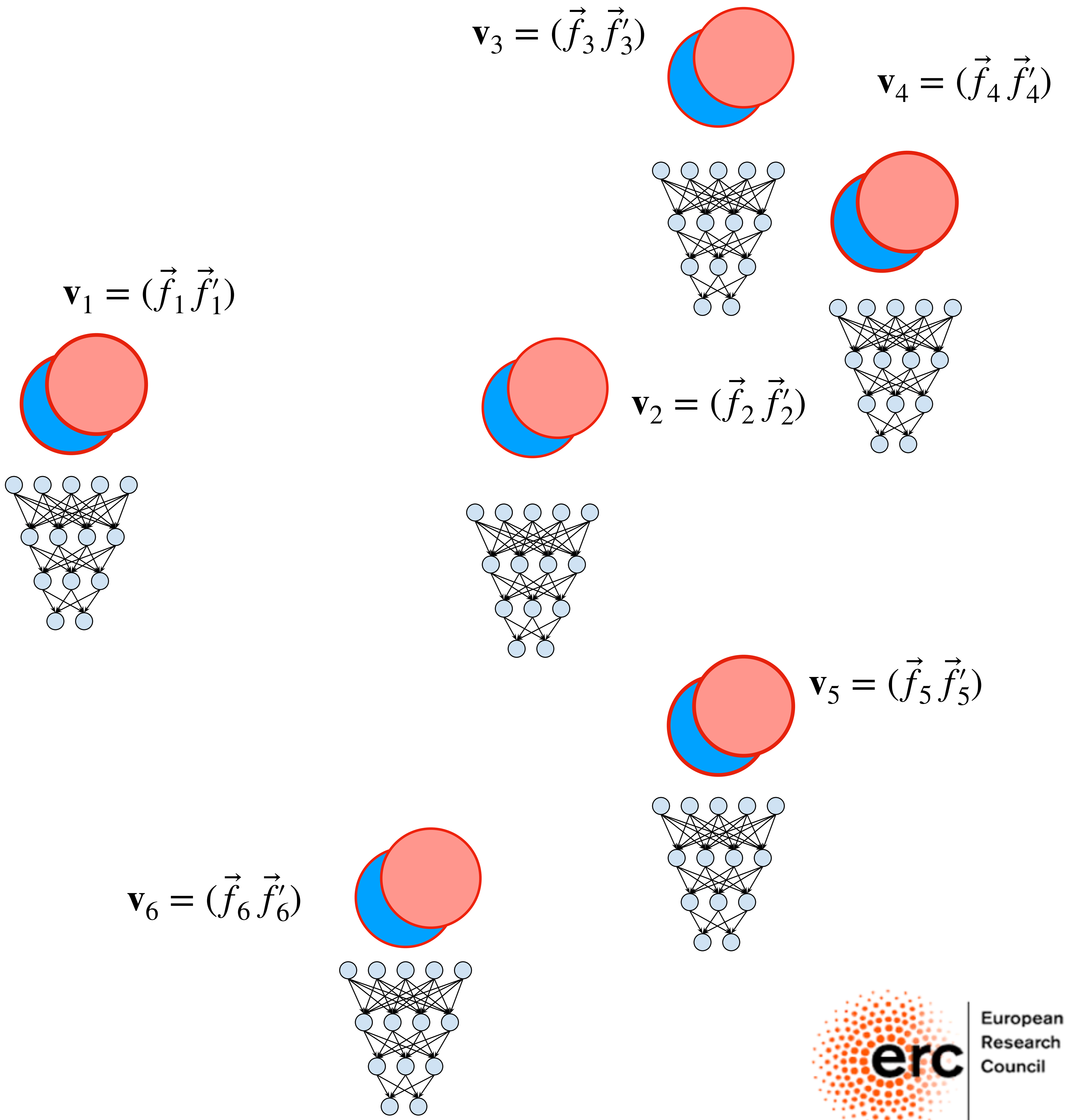
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - *Each example in a dataset is represented as a set of vertices*
  - *Each vertex is embedded in the graph as a vector of features*
  - *Vertices are connected through links (edges)*
  - *Messages are passed through 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

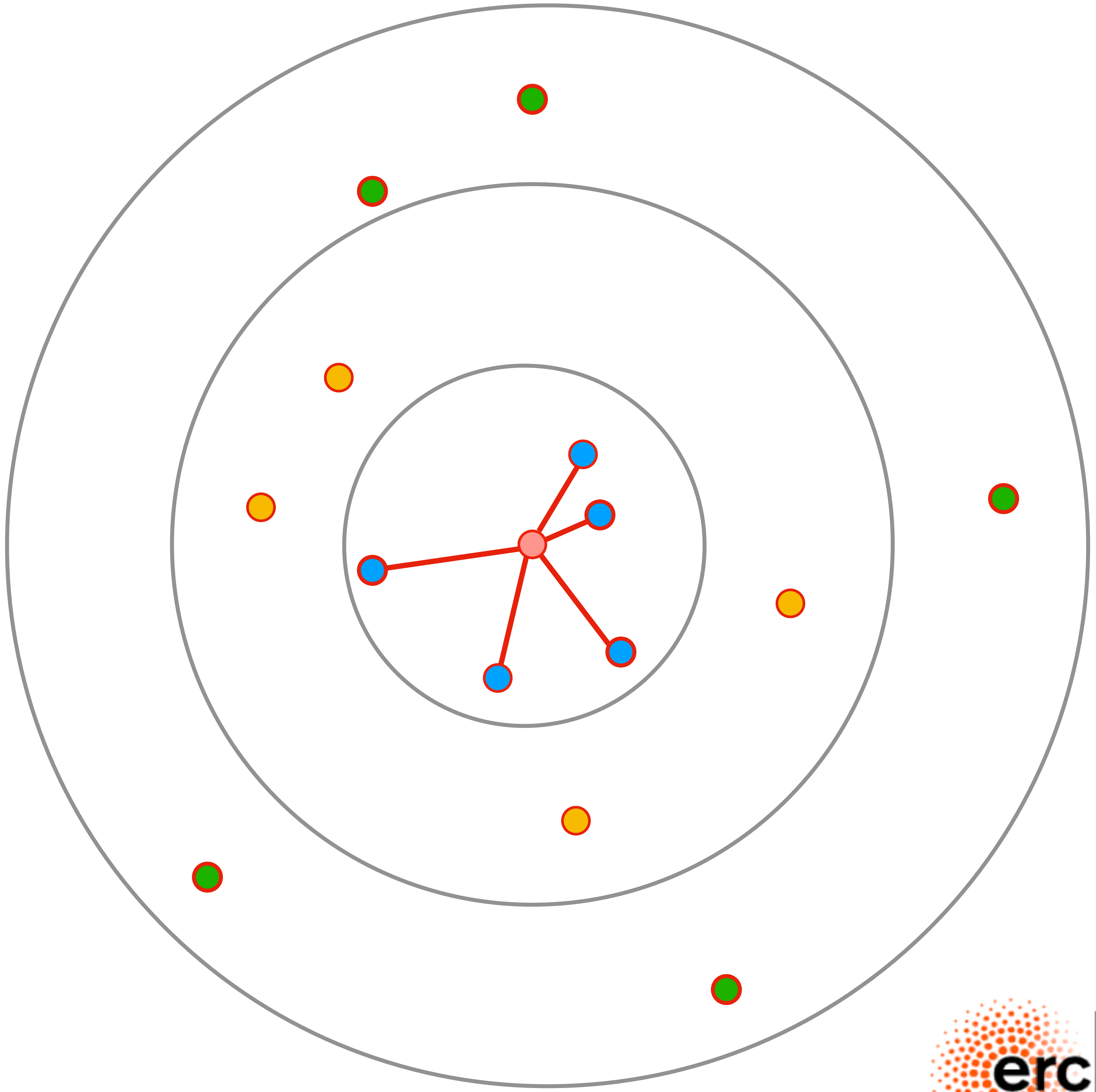
The inference step

- The inference step usually happens on each vertex
- But, depending on the problem, it might happen across the graph
- Usually, this is done with a DNN taking
    - the initial features f_i
    - the learned representation f_i'
    - [optional] some ground-truth label (for classifiers)



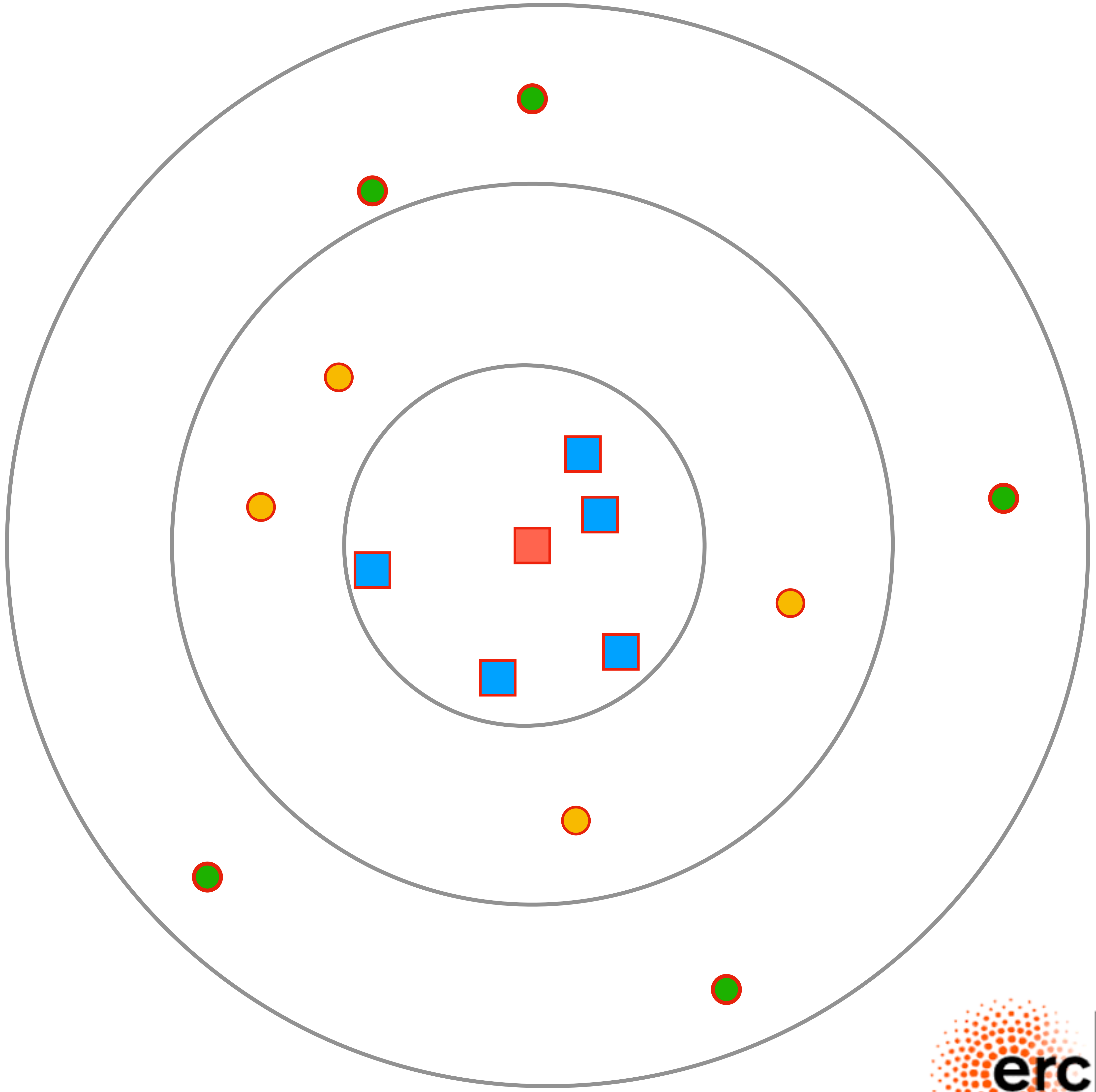
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



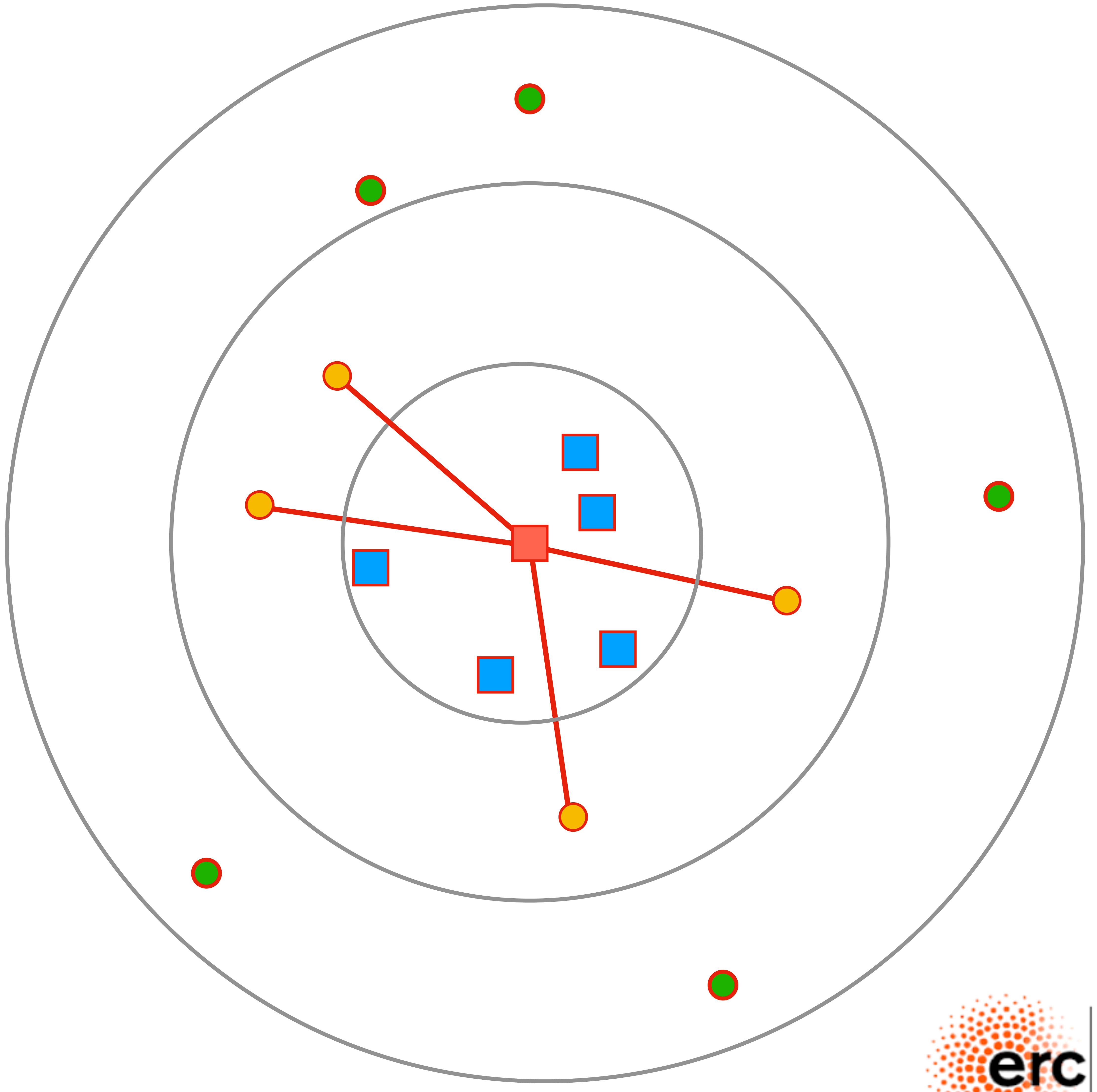
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



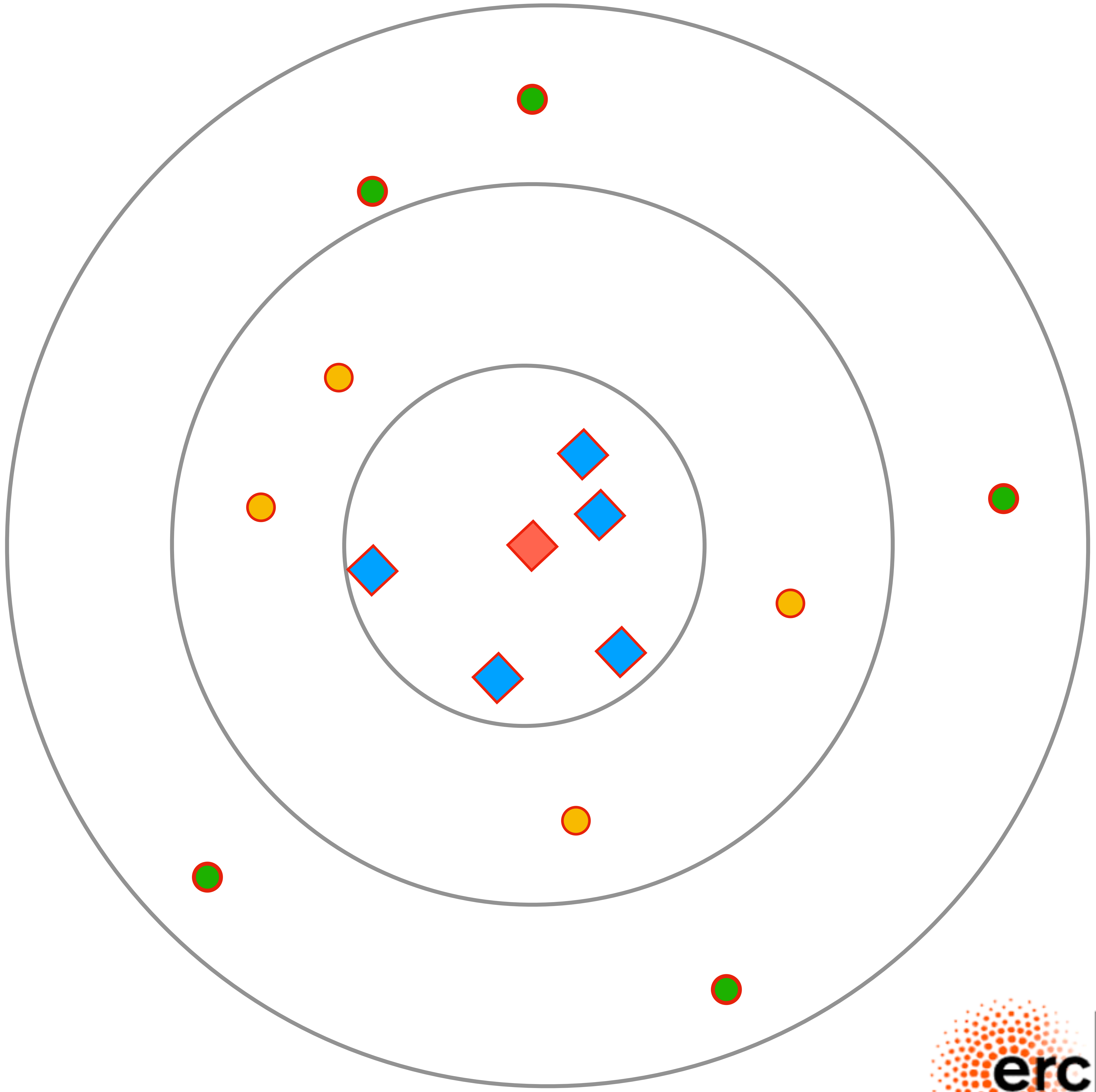
...and repeat

- Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex
  - You could start from coordinates in real space + some feature
  - Build function of them
  - Build functions of functions of them
  - At each step, you improve knowledge on your vertex V



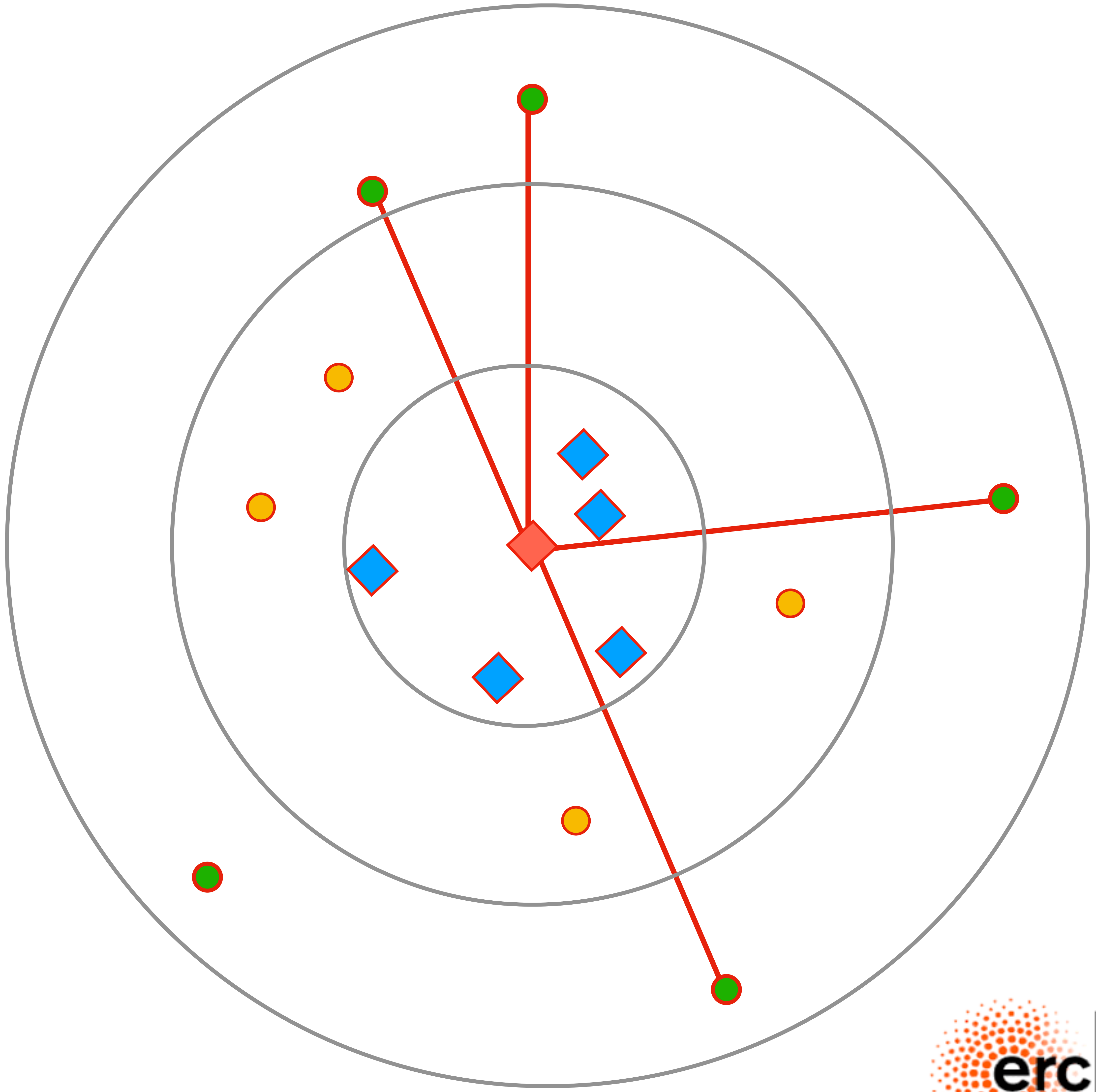
...and repeat

- Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex
  - You could start from coordinates in real space + some feature
  - Build function of them
  - Build functions of functions of them
  - At each step, you improve knowledge on your vertex V



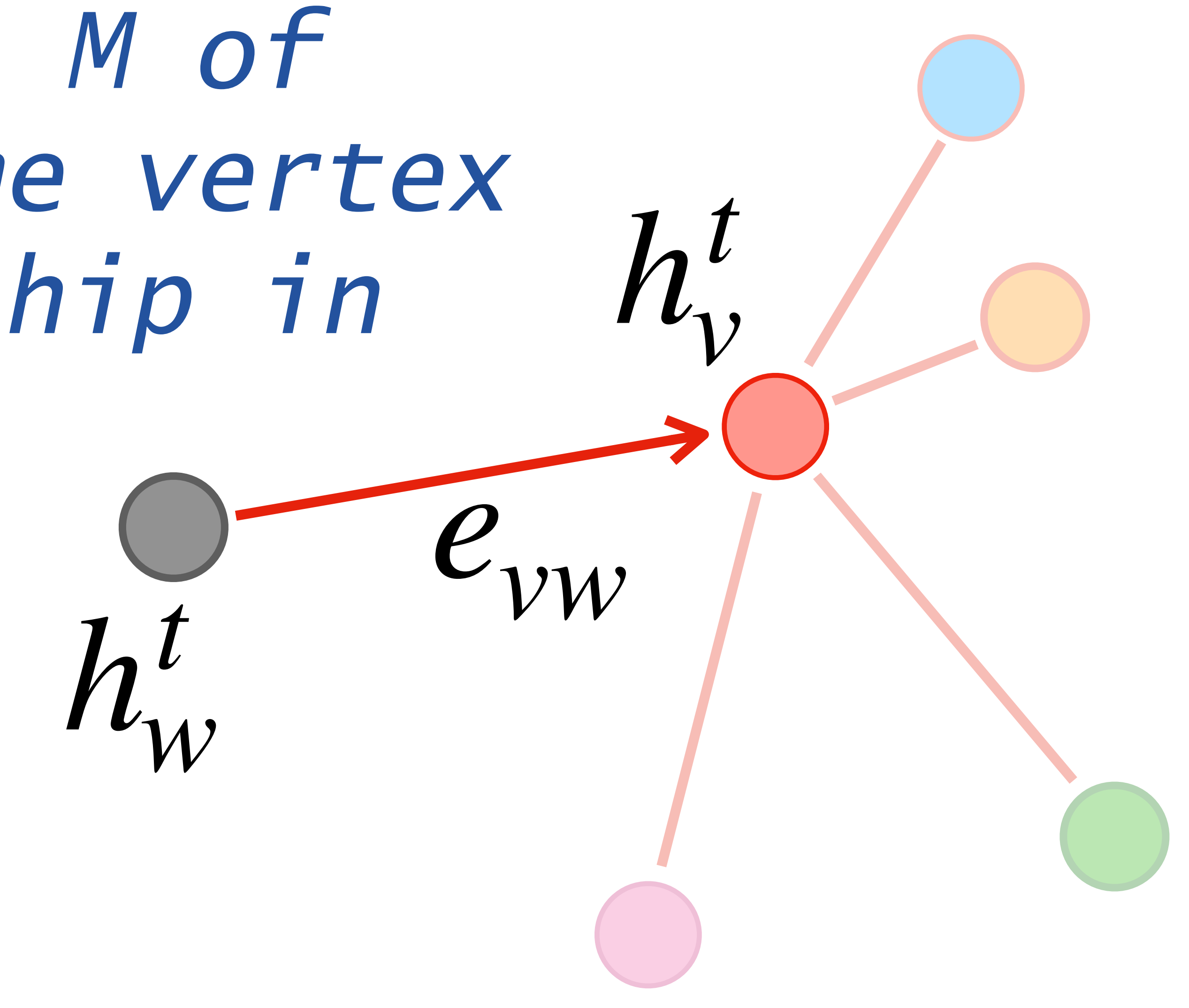
…and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



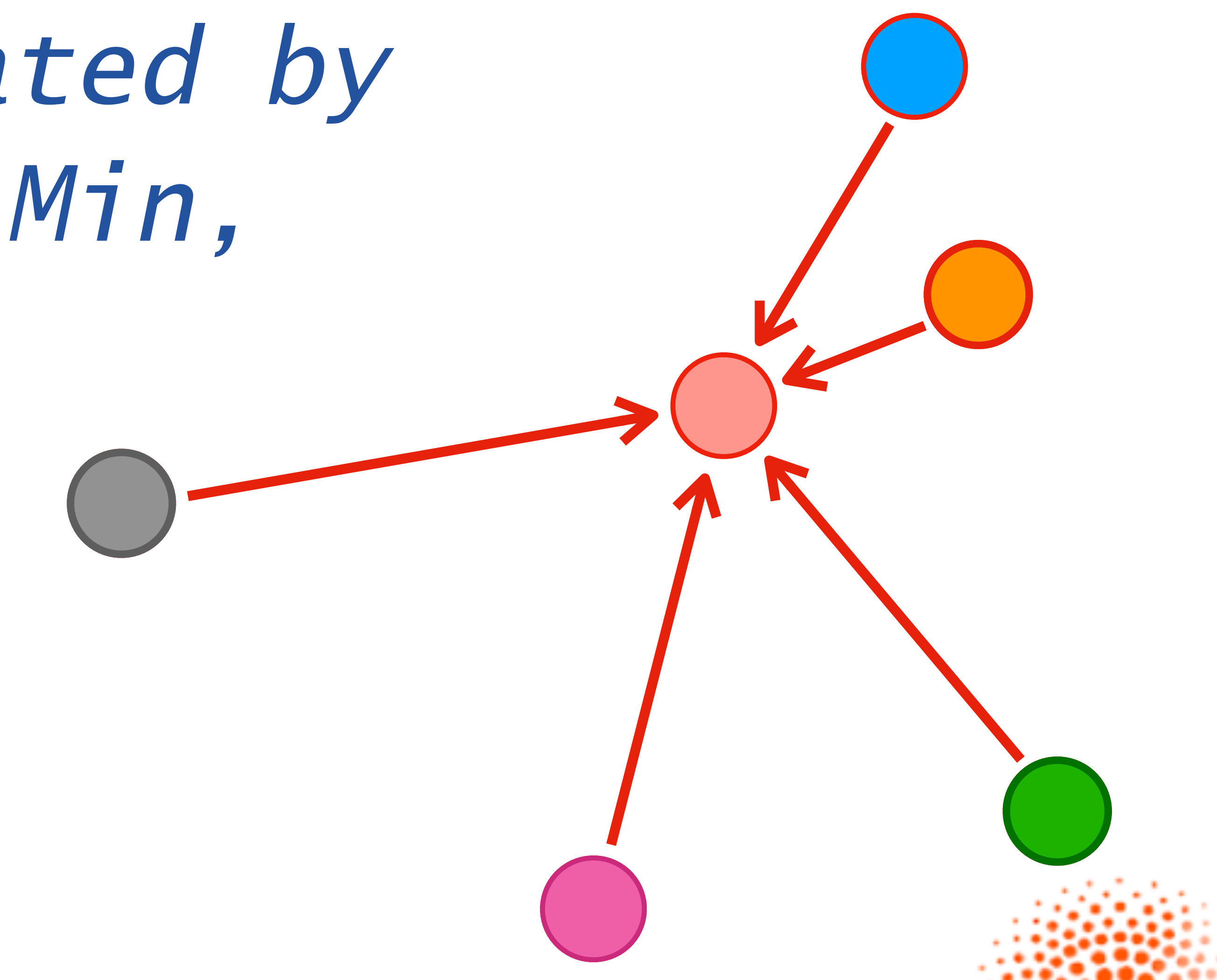
With equations…

- 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)



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

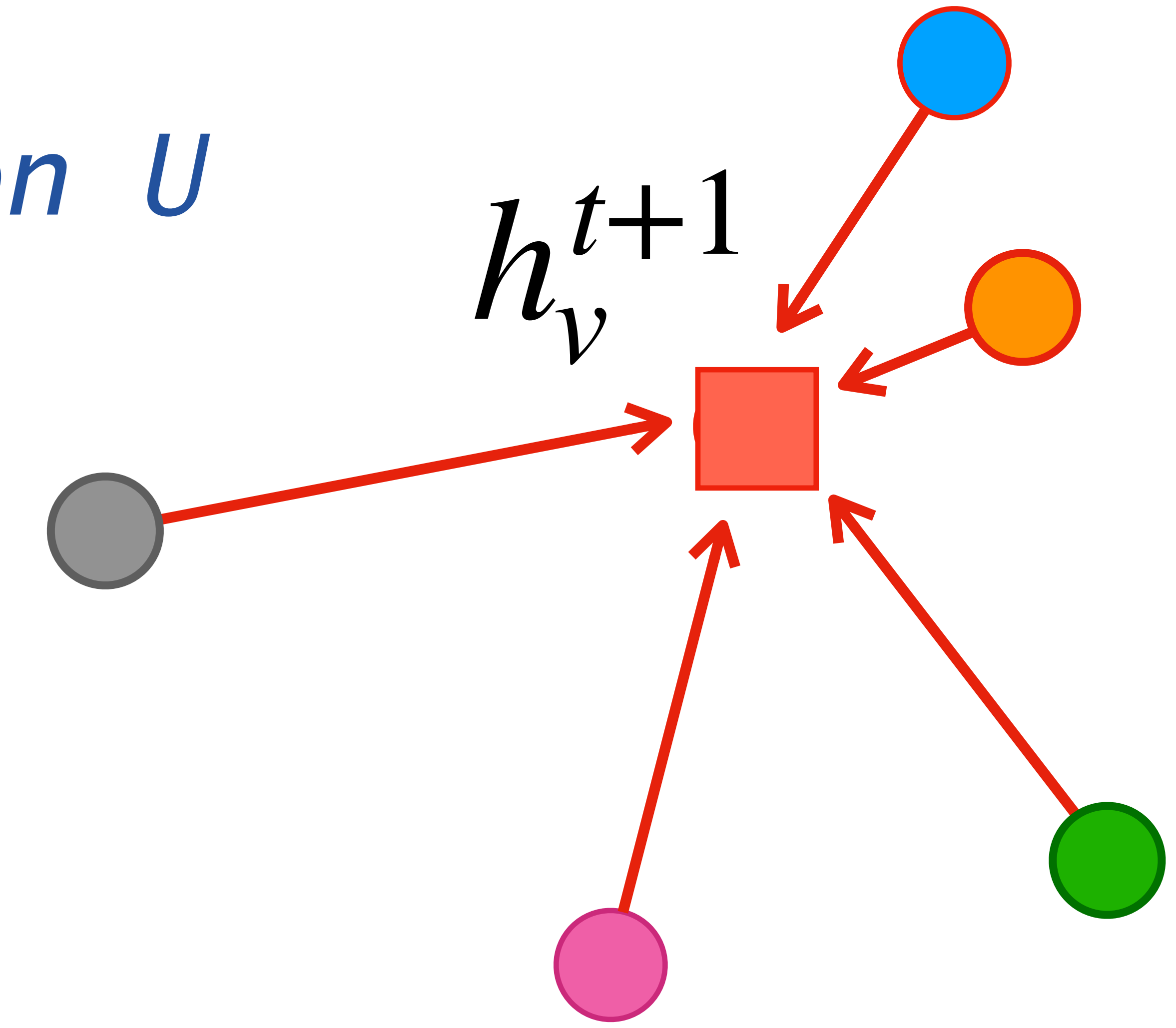
- The message carried to a vertex v is aggregated by some function (typically sum, but also Max, Min, etc.)



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

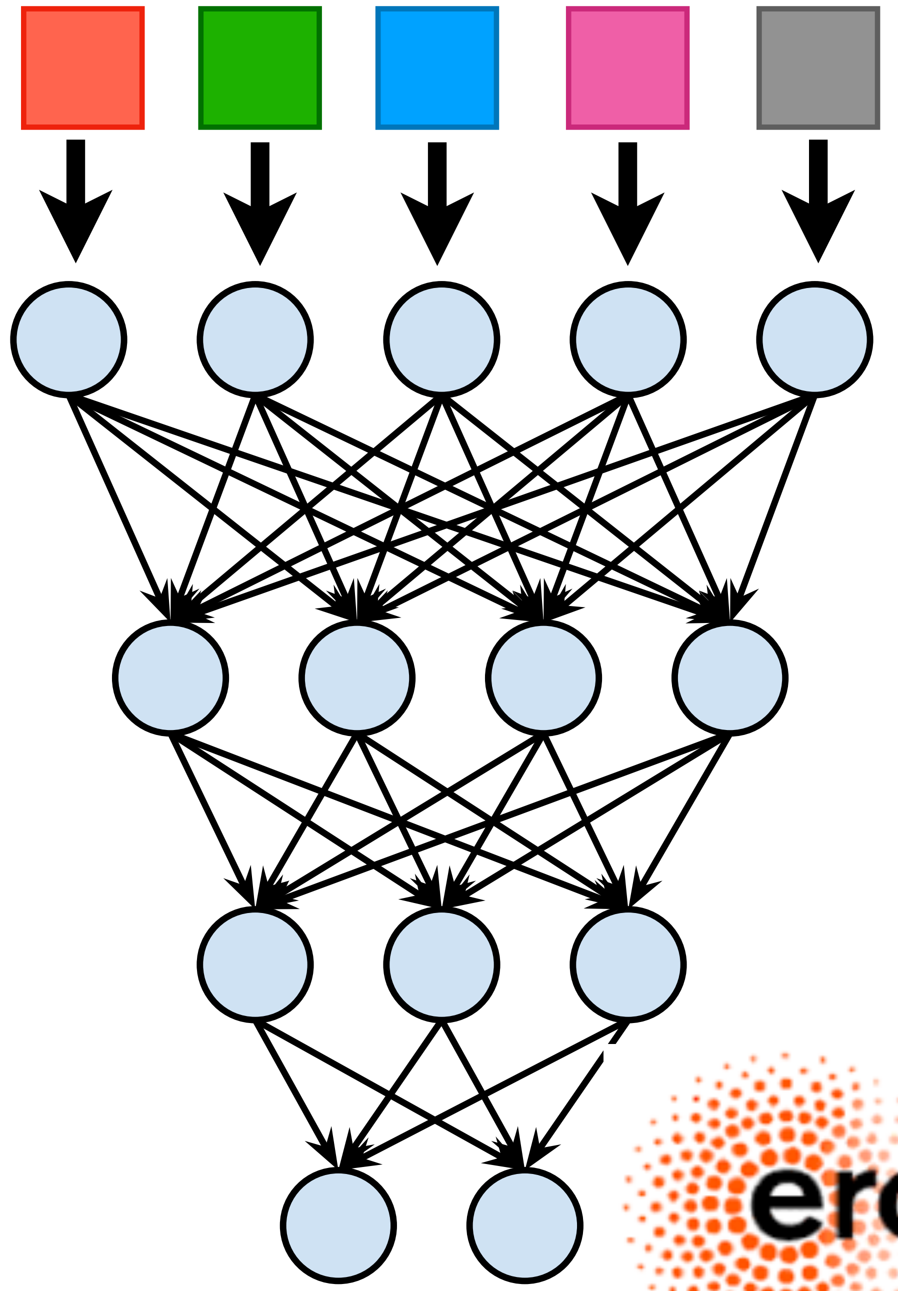
With equations...

- *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*



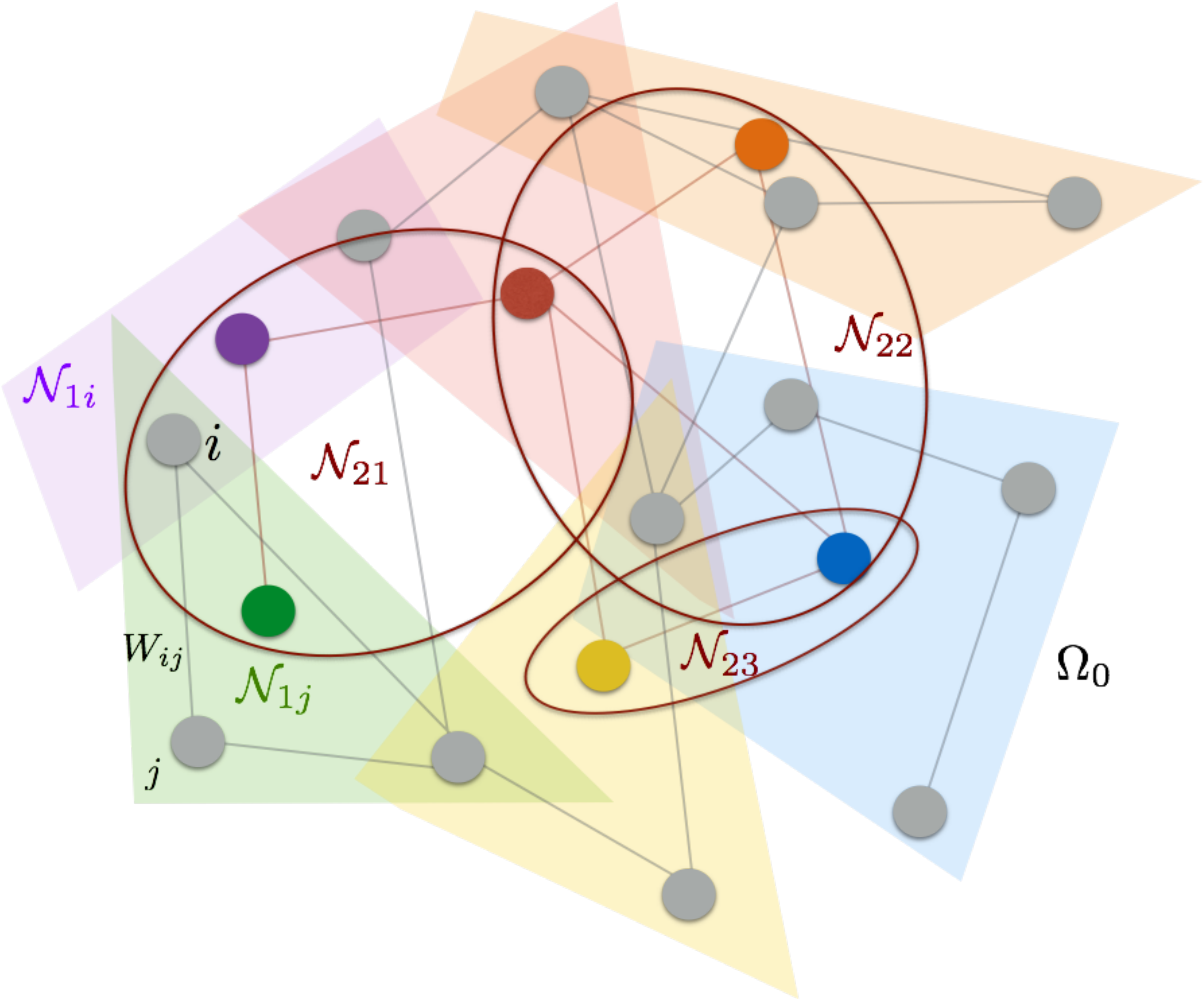
$$\hat{y} = R(h_v^T \,|\, v \in G)$$

Learning Message

- Typically, the M, U, and R functions are learned from data
  - Expressed as neural networks (fully connected NNs, recurrent NNs, etc.)
  - Which networks to use depends on the specific problem, as much as the graph-building rules
- But you could inject domain knowledge in the game
  - You might know that SOME message is carried by some specific functions (e.,g., Netwon's low for N-body system simulation)
  - You could then use analytic functions for some message
  - You could still use a learned function for other messages
- The trick is dealing with differentiable functions not to spoil your back propagation
  - Graph networks become a tool for probabilistic programming

A little bit of History

- *(in this millenium) Graph networks started (as often it is the case) with a Yann LeCun et al. paper*
- *They tried to generalise CNNs beyond the regular-array dataset paradigm*
- *They replaced the translation-invariant kernel structure of CNNs with hierarchical clustering*

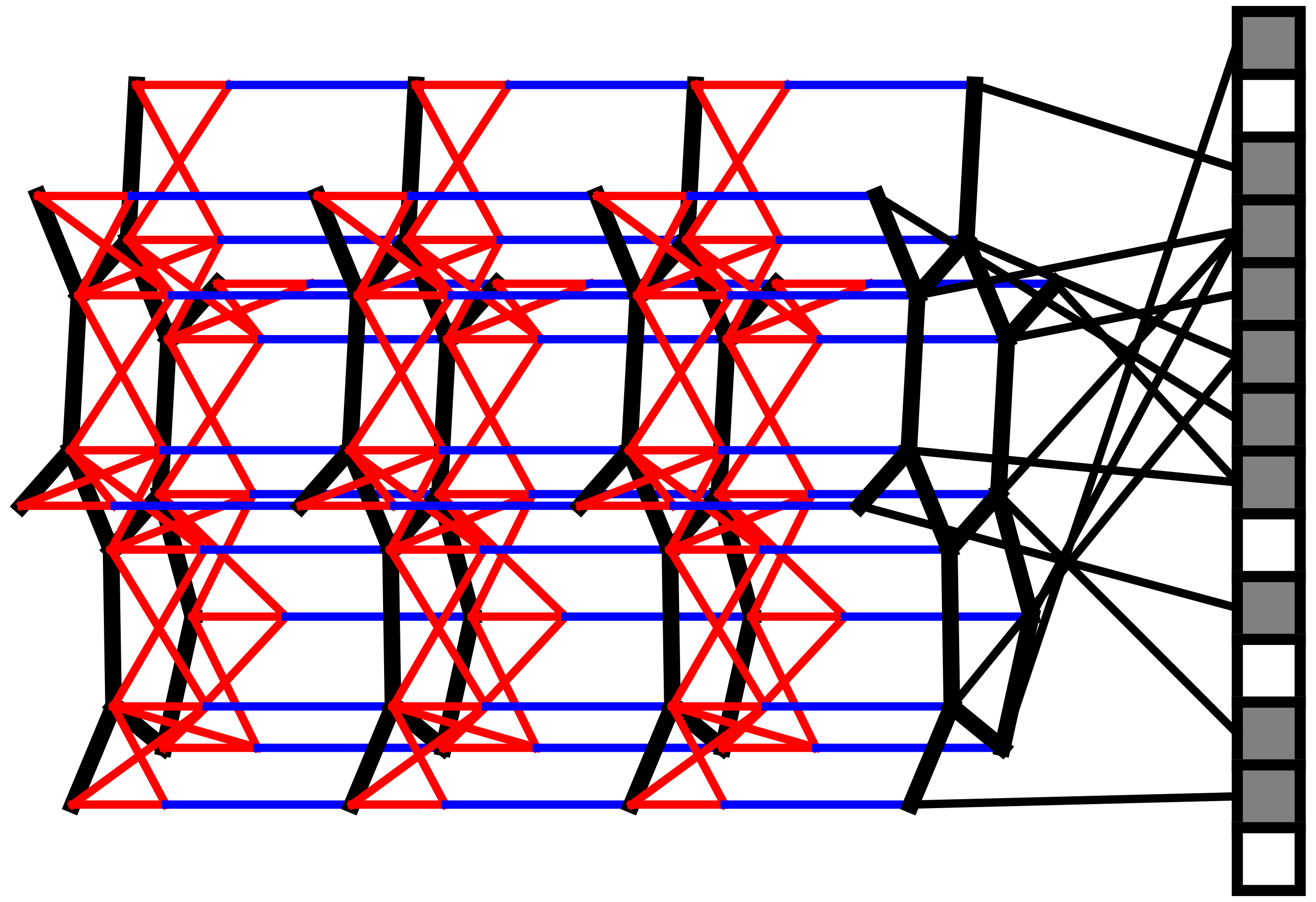


https://arxiv.org/abs/1312.6203

European Research Council

A little bit of History

- *The idea of message passing can be tracked to a ’15 paper by Duvenaud et al.*
- *The paper introduces “a convolutional neural network that operates directly on graphs”*
- *Language is different, but if you look at the algorithm it is pretty much what we discussed (for specific network architecture choices)*



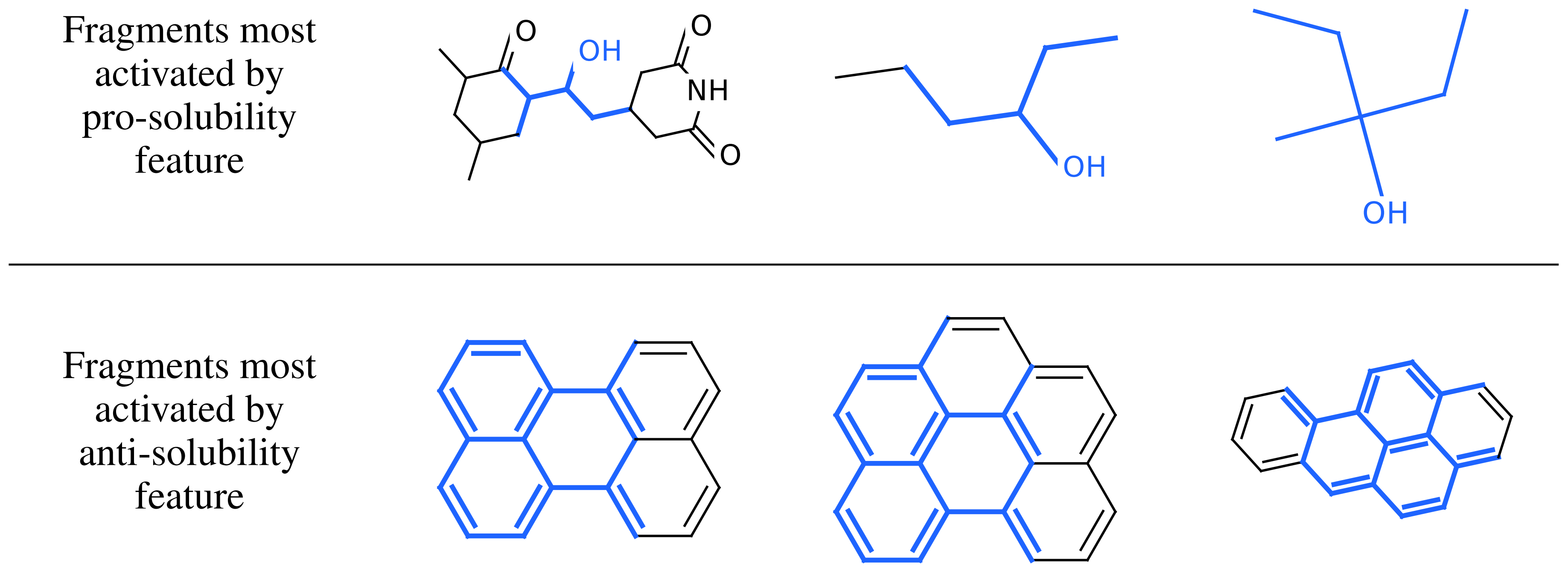


Figure 4: Examining fingerprints optimized for predicting solubility. Shown here are representative examples of molecular fragments (highlighted in blue) which most activate different features of the fingerprint. *Top row:* The feature most predictive of solubility. *Bottom row:* The feature most predictive of insolubility.

https://arxiv.org/pdf/1509.09292.pdf

Algorithm 2 Neural graph fingerprints

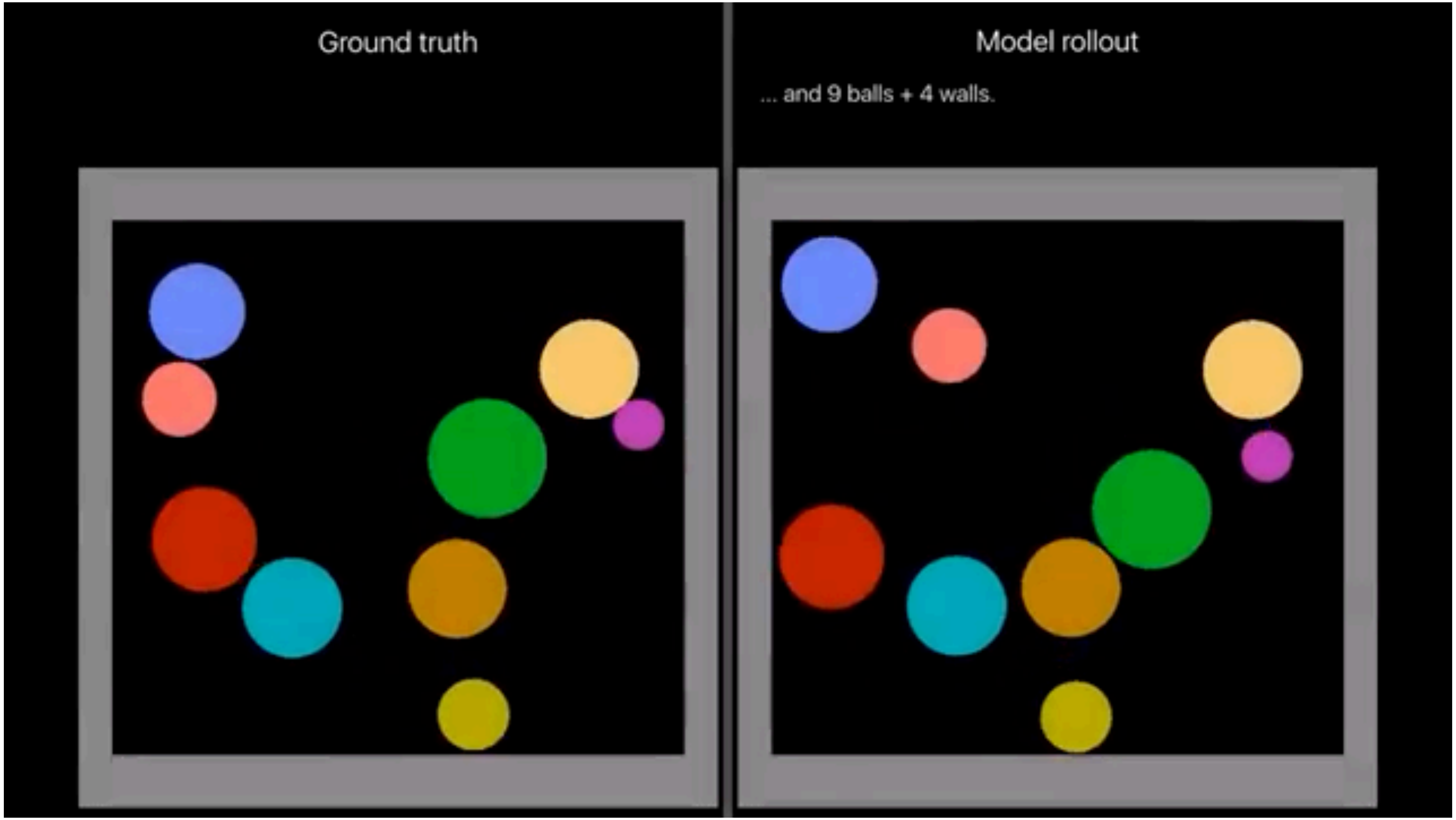
Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
    $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
    $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
    $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

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*
- *A few GitHub examples*
  - *JEDI-net Interaction Networks for jet tagging on these data*
  - *PUPPIML: GGNN for pileup subtraction*
  - *A small GarNet example that fits an FPGA on these data*

European Research Council

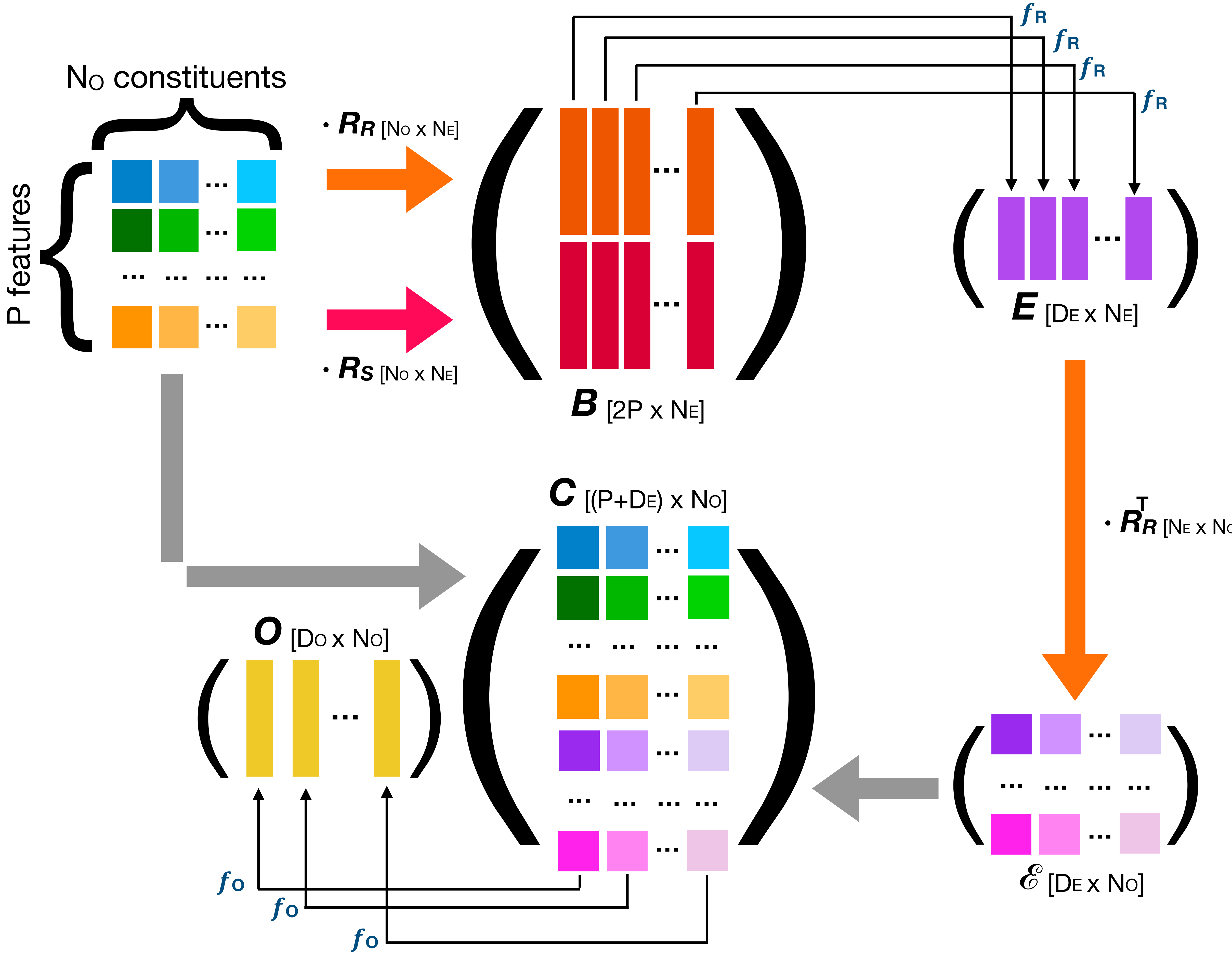
Interaction Networks



https://arxiv.org/abs/1612.00222

Interaction Networks

- *INs process a list of N_O x P inputs in pairs, through Receiving and Sending matrices*



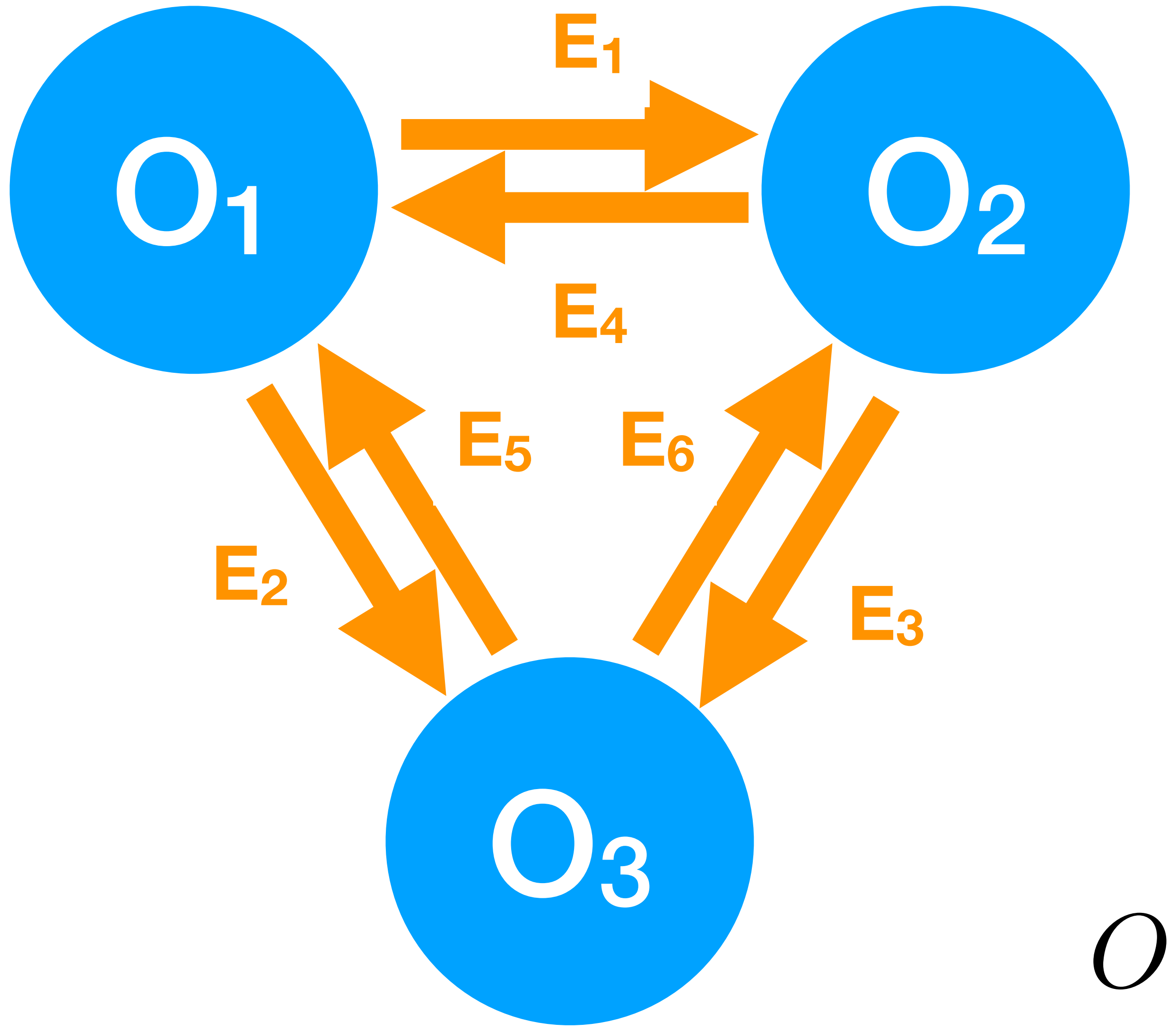
N_O: # of constituents
P: # of features
N_E = N_O(N_O-1): # of edges
D_E: size of internal representations
D_O: size of post-interaction internal representation

ϕ_C, f_O, f_R parameterized as neural networks

Interaction Networks

- *INs process a list of No x P inputs in pairs, through Receiving and Sending matrices*

- *The effect of the interaction is learned by fR and combined with the input to learn (through fo) a post-interaction representation*

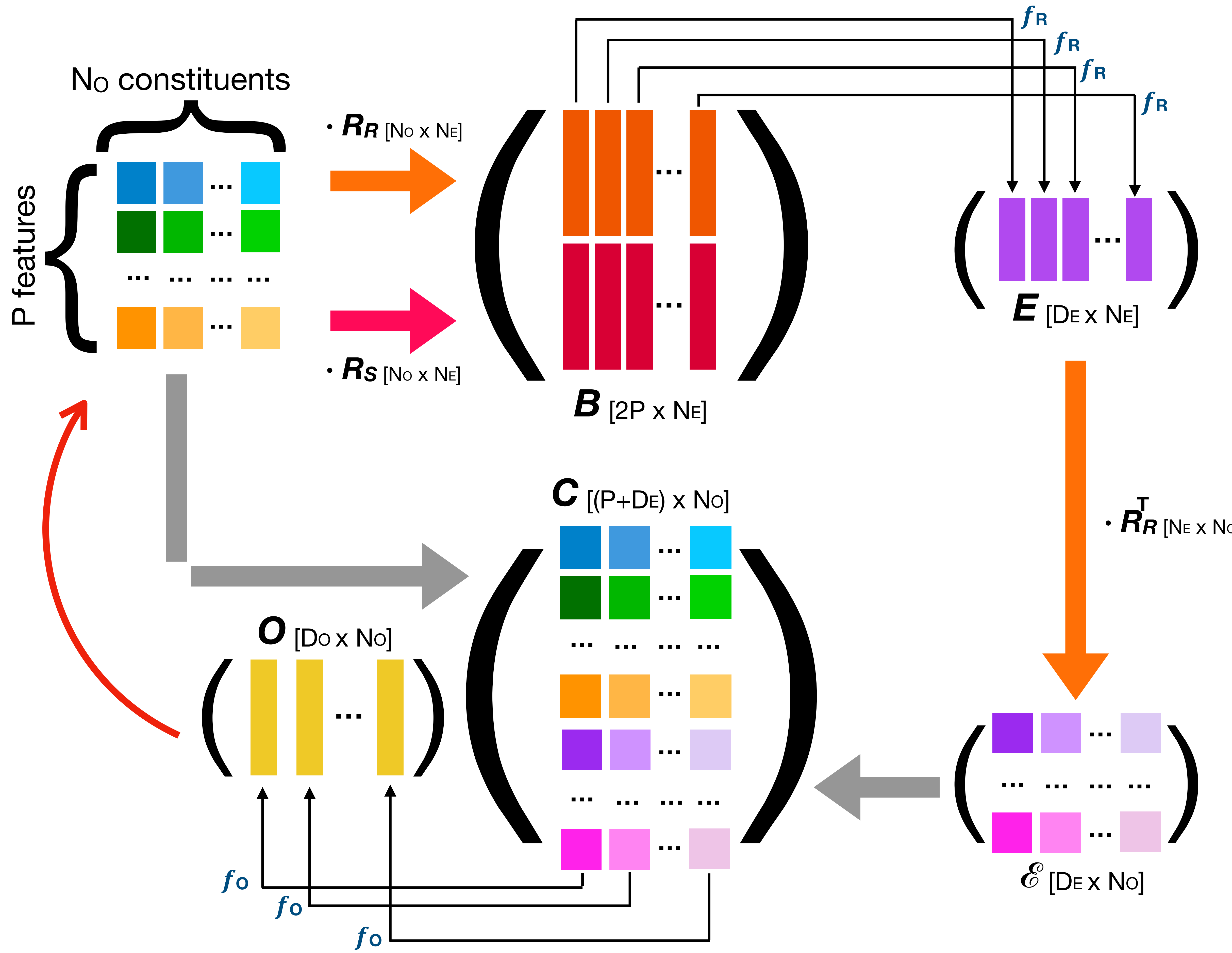


$$
R_R = \begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \end{array}
\begin{array}{c}
\begin{array}{cccccc} E_1 & E_2 & E_3 & E_4 & E_5 & E_6 \end{array} \\
\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}
\end{array}
$$

$$
R_S = \begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \end{array}
\begin{array}{c}
\begin{array}{cccccc} E_1 & E_2 & E_3 & E_4 & E_5 & E_6 \end{array} \\
\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}
\end{array} .
$$

Interaction Networks

- INs process a list of No x P inputs in pairs, through Receiving and Sending matrices
- The effect of the interaction is learned by fR and combined with the input to learn (through fo) a post-interaction representation
- The procedure can then be *iterated* to produce further steps i the interactions



NO: # of constituents
P: # of features
NE = NO(NO-1): # of edges
DE: size of internal representations
DO: size of post-interaction internal representation

ϕ_C, f_O, f_R parameterized as neural networks

With equations

- *The input is a vector, obtained concatenating sender and receiver feature*

$$\overrightarrow{x^{sr}} = (\overrightarrow{x^{s}}, \overrightarrow{x^{r}})$$

- *The input is processed by a network, that compute "kernel" functions of these inputs*

$$f_R^{sr}(\overrightarrow{x^{sr}})$$

- *Message across senders is gathered by summing*

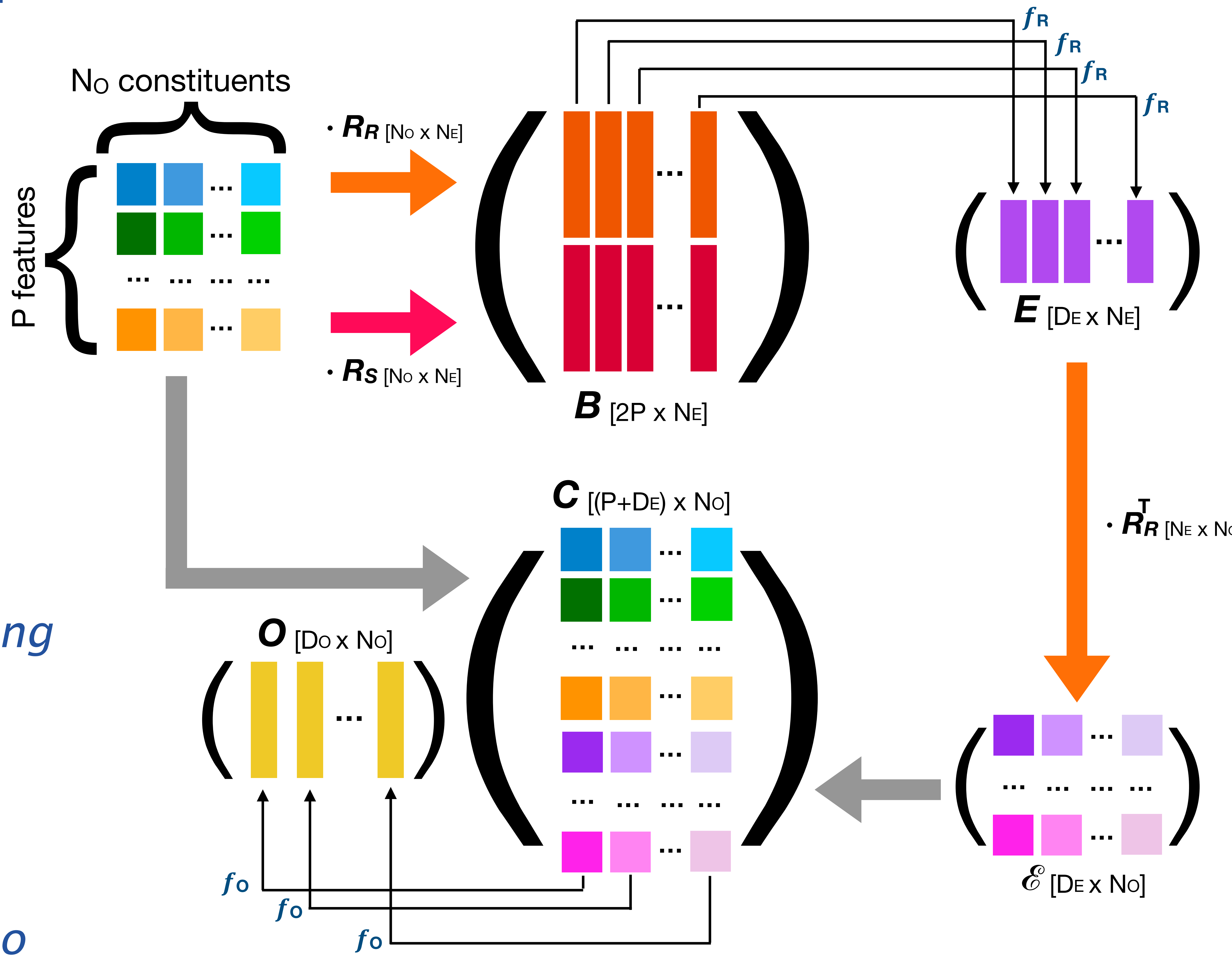
$$\overrightarrow{e^{r}} = \sum_{s} f_R^{sr}(\overrightarrow{x^{sr}})$$

- *The interaction features are concatenated to the input*

$$\overrightarrow{c^{r}} = (\overrightarrow{e^{r}}, \overrightarrow{x^{r}})$$

- *A final neural network returns the post-interaction representation*

$$\overrightarrow{o^{r}} = f_O(\overrightarrow{c^{r}})$$



European Research Council

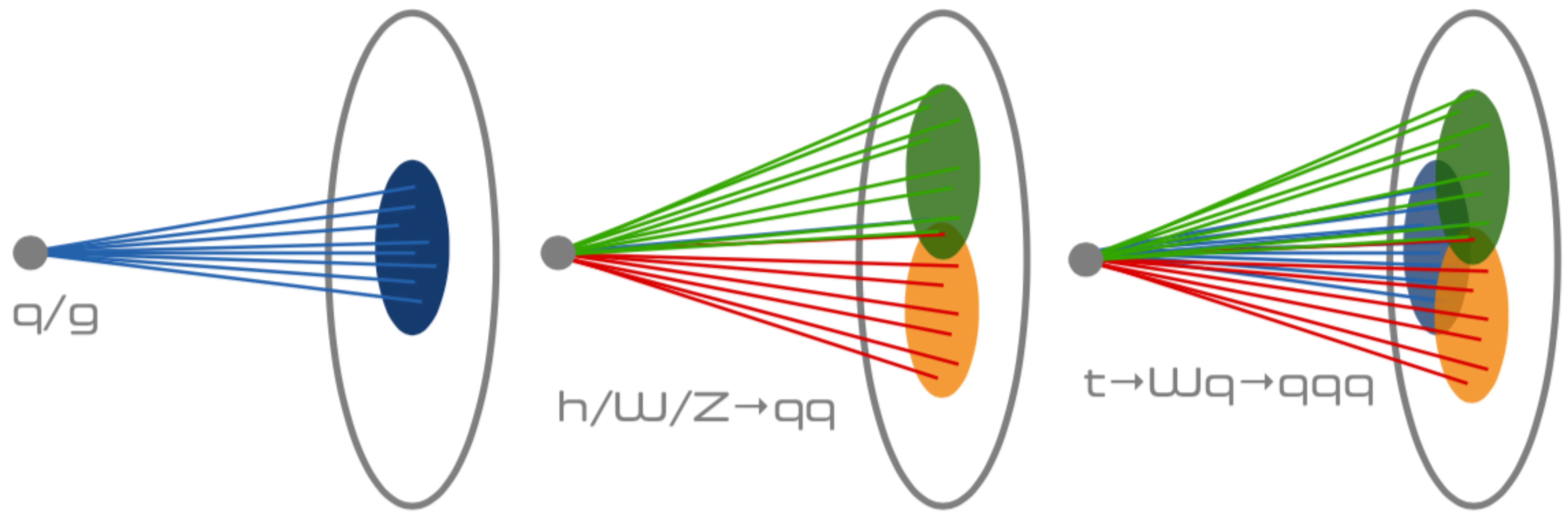
Example: jet tagging

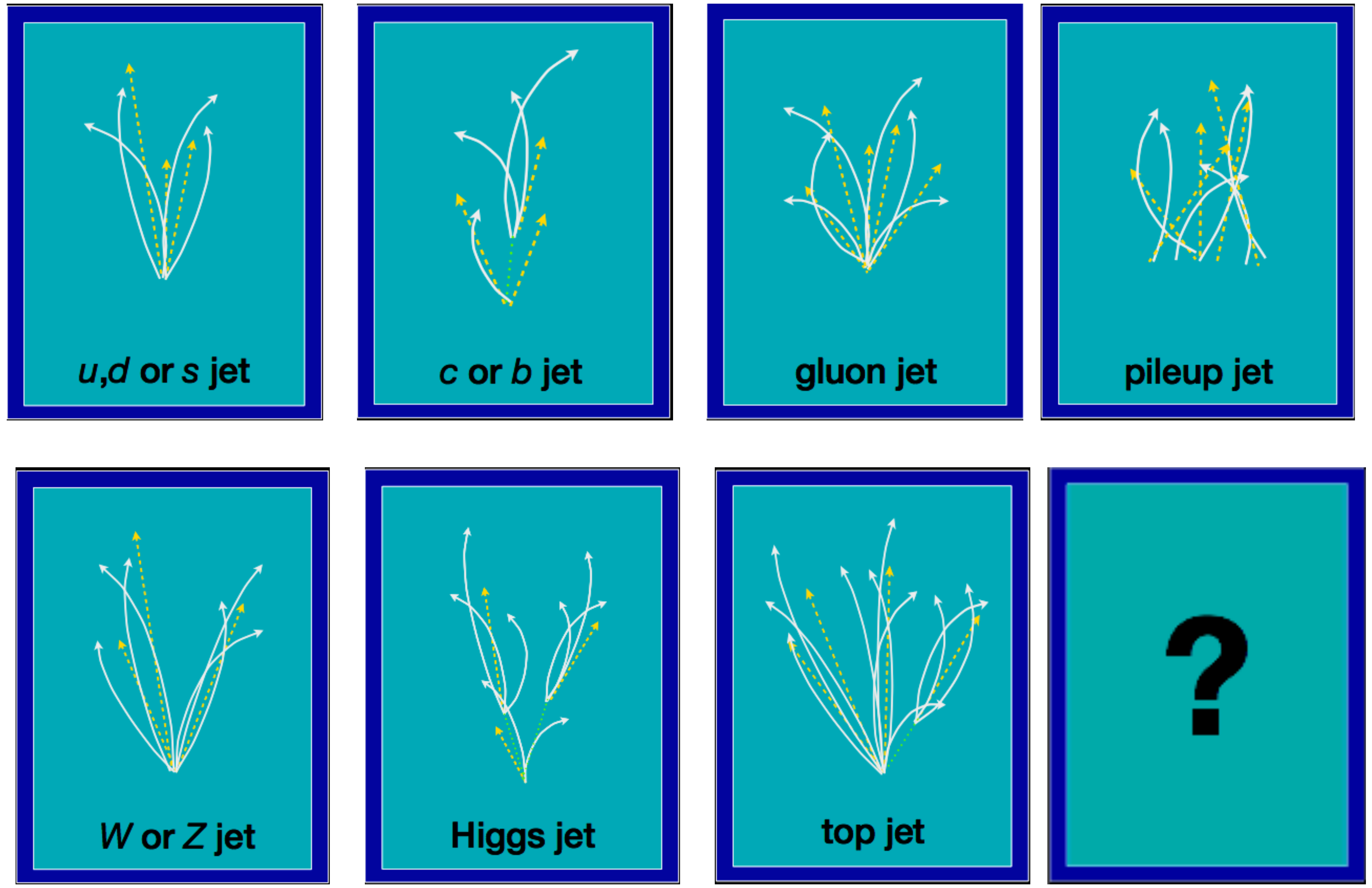
- You have a jet at LHC: spray of hadrons coming from a "shower" initiated by a fundamental particle of some kind (quark, gluon, W/Z/H bosons, top quark)

- You have a set of jet features whose distribution depends on the nature of the initial particle

- You can train a network to start from the values of these quantities and guess the nature of your jet

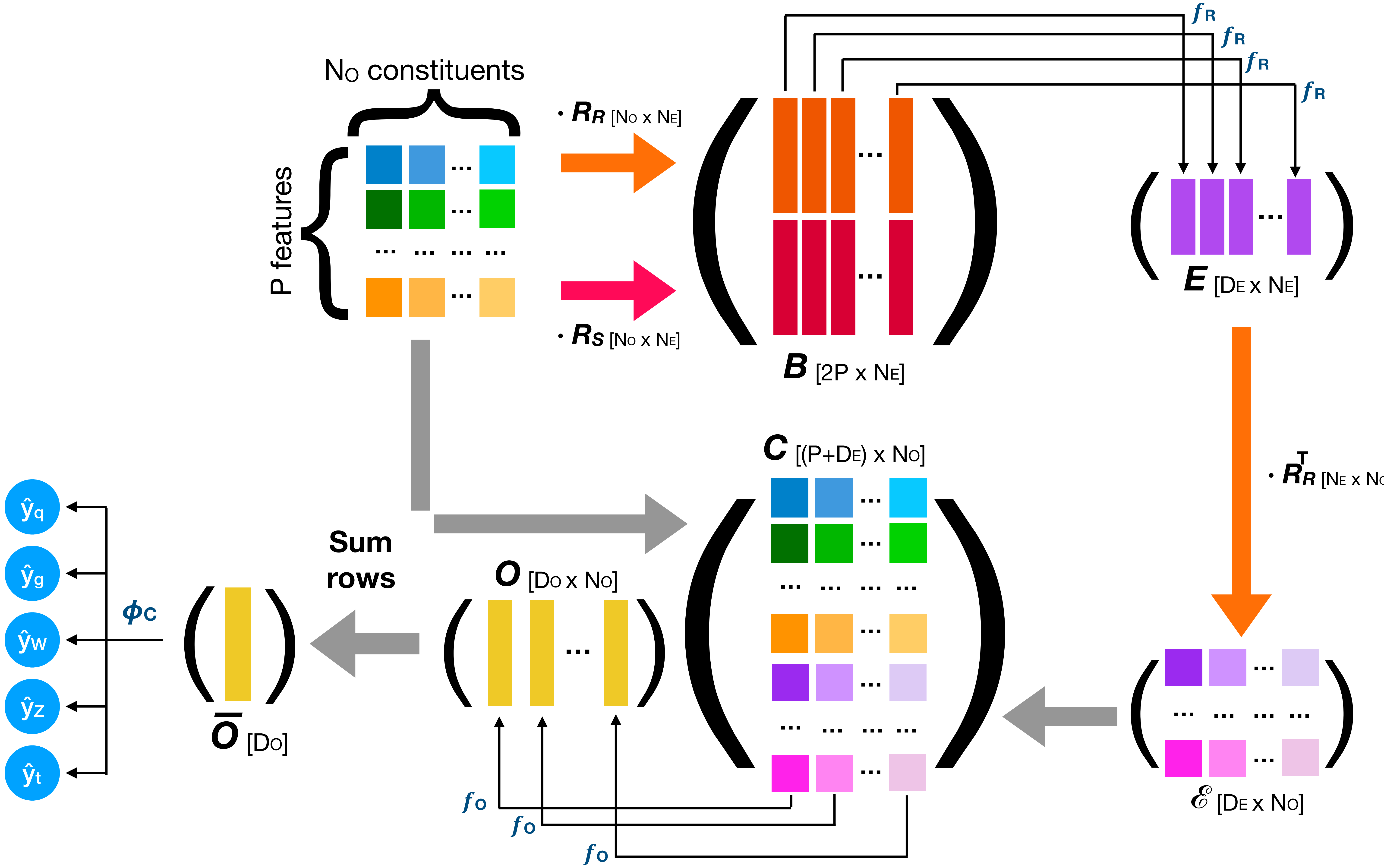
- To do this you need a sample for which you know the answer





INs for Jet Identification

- *In this case, there is no system update needed (i.e., no cycle)*
- *It is sufficient to use the post-interaction representation as input to a classifier that returns the jet category*
- *The three networks are simultaneously optimized: the learned representation is chosen to help the classification*

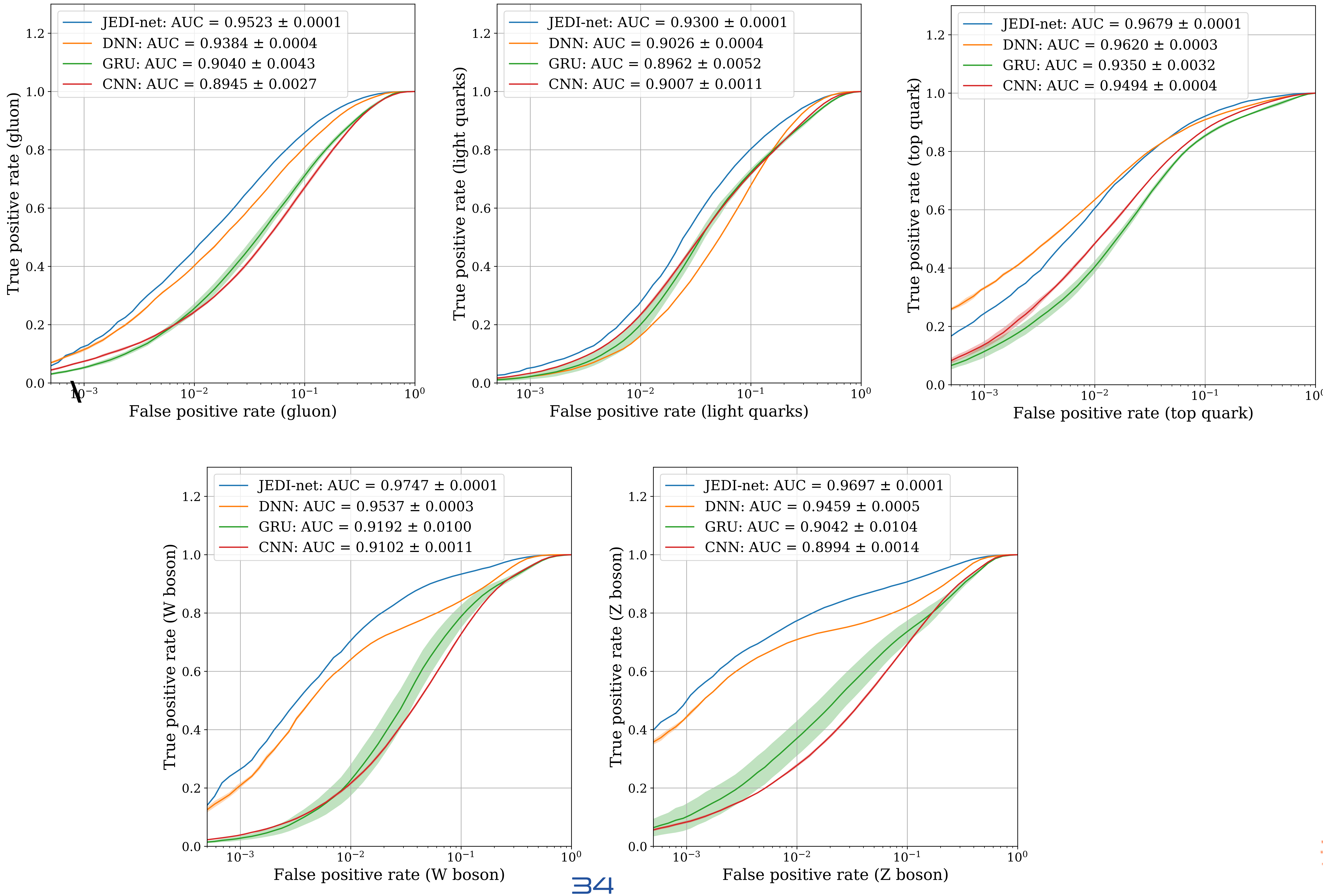


N_O: # of constituents
P: # of features
$N_E = N_O(N_O-1)$: # of edges
D_E: size of internal representations
D_O: size of post-interaction internal representation

ϕ_C, f_O, f_R parameterized as neural networks

European Research Council

A comparison



Summary

- *Graph Networks are a powerful tool to learn from sparse data sets*
  - *extend CNN concept beyond the case of geometrical proximity -> learned representation*
  - *allow to abstract from irregular geometry (molecules, particle-physics detectors, stars in a galaxy, …)*
  - *allow to inject domain knowledge in the game (e.g., enforcing physics rules for message-passing functions [Newton's law in N-body simulation]*
  - *But can also be used to learn (how to simulate) physics*