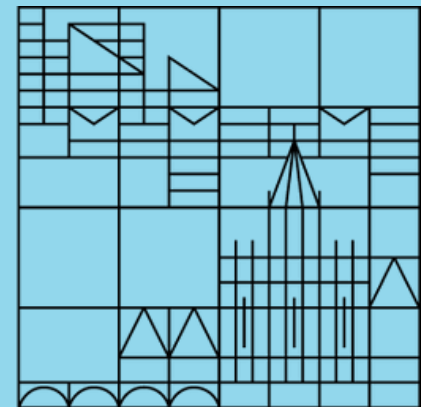


Universität
Konstanz



UNIVERSITÄT KONSTANZ

Network Science Advanced Topics

Network Science of
Socio-Economic Systems
Giordano De Marzo

Recap

Polarization and Online Echo Chambers

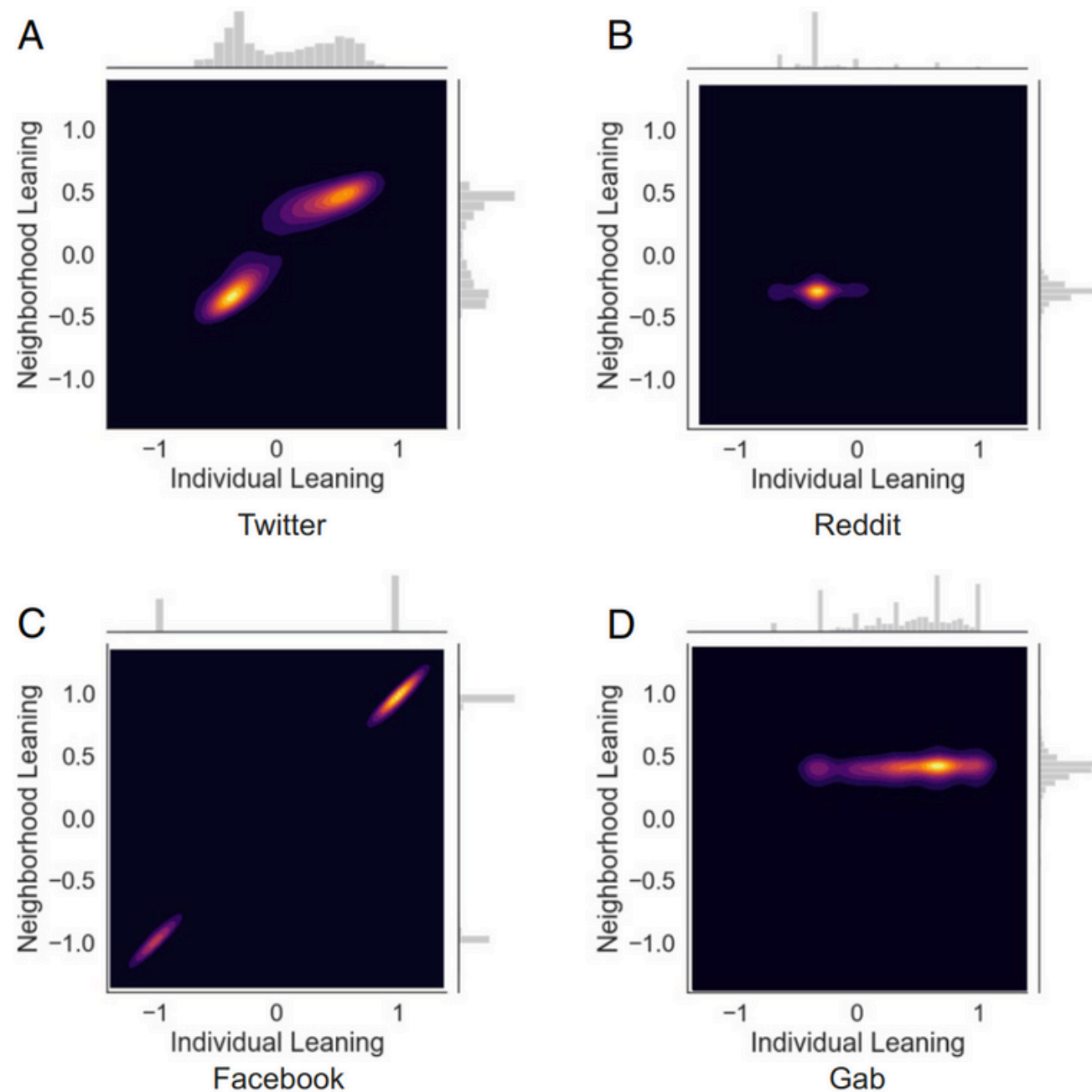
Increasing levels of (multidimensional) polarization are observed in society. This phenomenon is stronger on social networks.

Spreading of (Mis)Information

Misinformation and disinformation are a serious threat to democracy. False news and rumors tend to spread more rapidly and deeply on online platforms.

Online Debunking

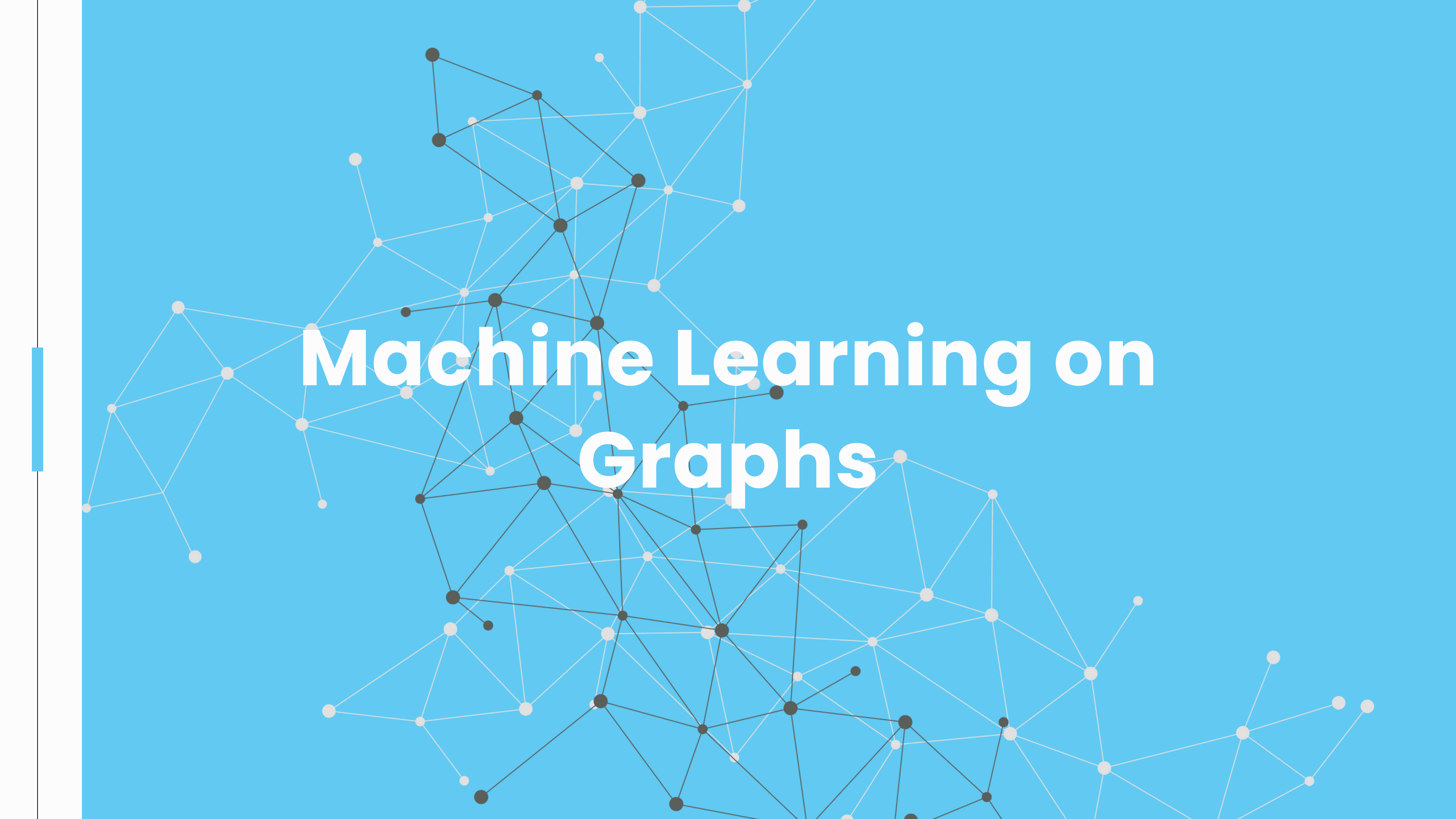
Studies show that standard fact-checking posts have little effect and, if any, they tend to worsen the situation.



Outline

1. Machine Learning on Graphs
2. Graph Neural Networks
3. Fitness Centrality



The background features a complex network graph with numerous nodes and edges. The nodes are represented by small circles, some of which are black and others are light gray. The edges are thin lines connecting these nodes, forming a dense web of connections. The overall aesthetic is clean and modern, typical of a technical or academic presentation.

Machine Learning on Graphs

Why Graphs are Important

In many tasks knowing the graph structure is crucial for capturing the functioning and properties of a system

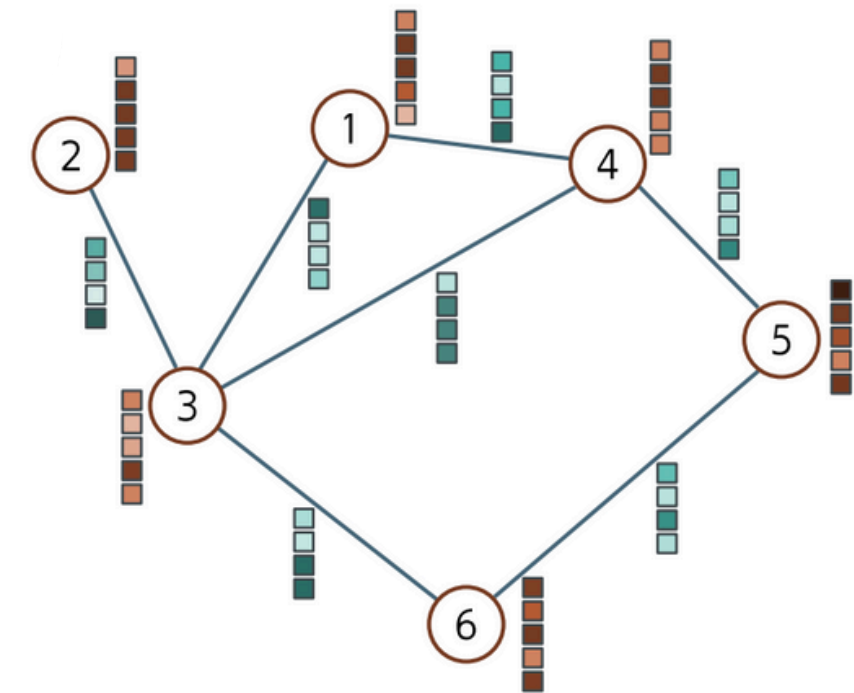
- the chemical properties of a molecule strongly depend on their structure. Knowing the chemical formula is generally completely unuseful
- on a social network, the behavior of a person can be strongly influenced by its friends. Often the features of the majority of people in a social group are more relevant than the individual features
- in order to perform recommendation of followers or friends, it is useful to take into account the social circle. The friend of my friend is more likely to be, in its turn, a friend of mine I haven't connected with yet

Machine Learning on Graphs allows to capture the graph structure in the data.

Graph Structured Data

Graph structured data are generally described by three distinct matrices:

- **Adjacency Matrix** Contains the network structure (following links on Instagram)
 - Size is $N \times N$
- **Node Data Matrix** Contains the D nodes features (gender, age, number of followers)
 - Size in $D \times N$
- **Edge Data Matrix** Contains the D_e edges features (when the following was made)
 - Size is $D_e \times E$



Adjacency matrix, A
 $N \times N$

	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	1	1	0	1	0	1
4	1	0	1	0	0	1
5	0	0	0	0	0	1
6	1	0	1	1	0	0

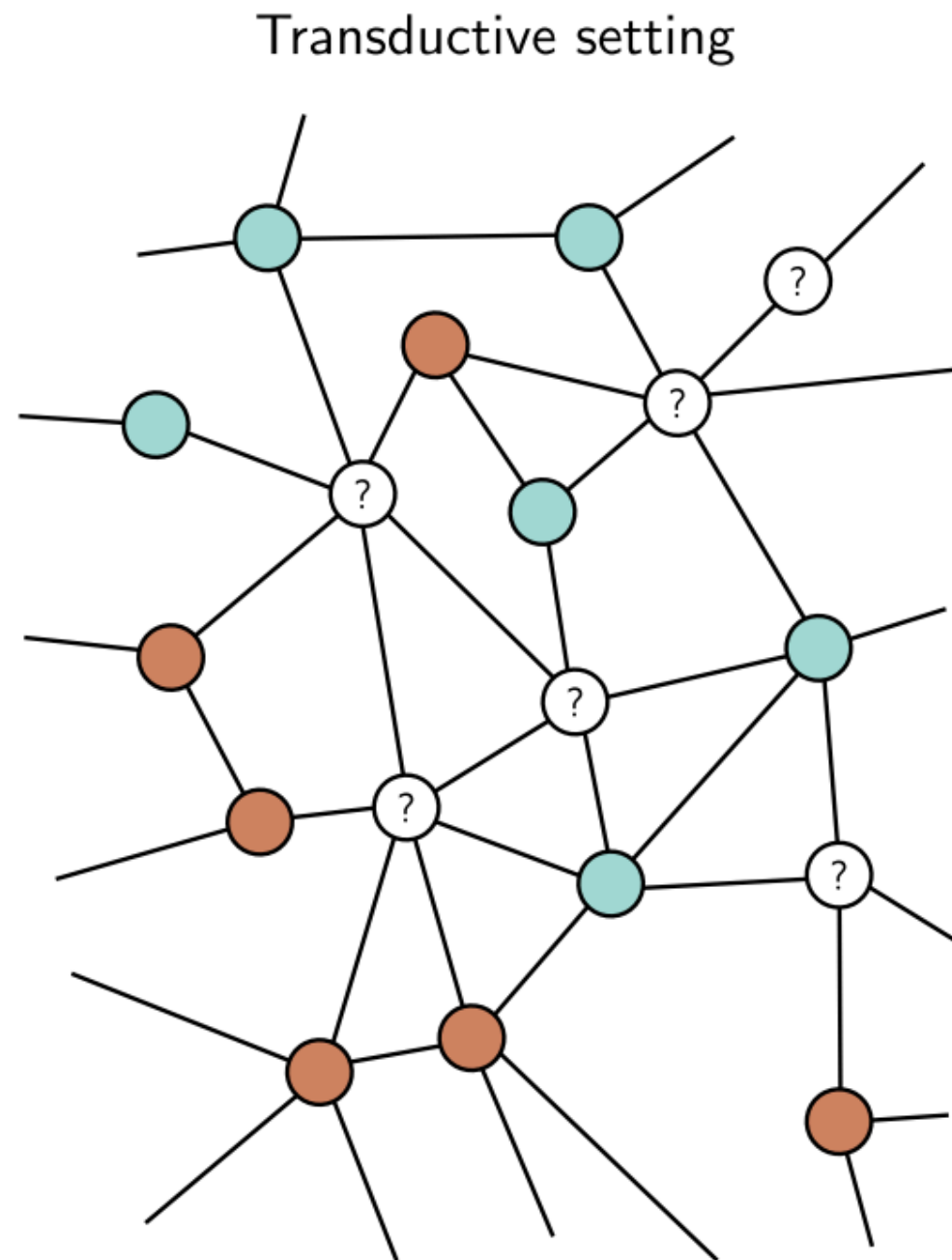
Node data, X
 $D \times N$

	1	2	3	4	5	6
1	0.8	0.7	0.6	0.5	0.4	0.3
2	0.9	0.8	0.7	0.6	0.5	0.4
3	0.7	0.6	0.5	0.4	0.3	0.2
4	0.6	0.5	0.4	0.3	0.2	0.1
5	0.5	0.4	0.3	0.2	0.1	0.0
6	0.4	0.3	0.2	0.1	0.0	0.0

Edge data, E
 $D_e \times E$

	1	1	2	3	3	4	5
	3	4	3	4	6	5	6
1	0.8	0.7	0.6	0.5	0.4	0.3	0.2
2	0.9	0.8	0.7	0.6	0.5	0.4	0.3
3	0.7	0.6	0.5	0.4	0.3	0.2	0.1
4	0.6	0.5	0.4	0.3	0.2	0.1	0.0
5	0.5	0.4	0.3	0.2	0.1	0.0	0.0
6	0.4	0.3	0.2	0.1	0.0	0.0	0.0

Node Classification



- As an example we consider a node classification task in a transductive setting:
- we only have one large graph
 - some of its nodes are labelled (train set)
 - the rest of the nodes have no label (test set)
 - the goal is to classify the test nodes

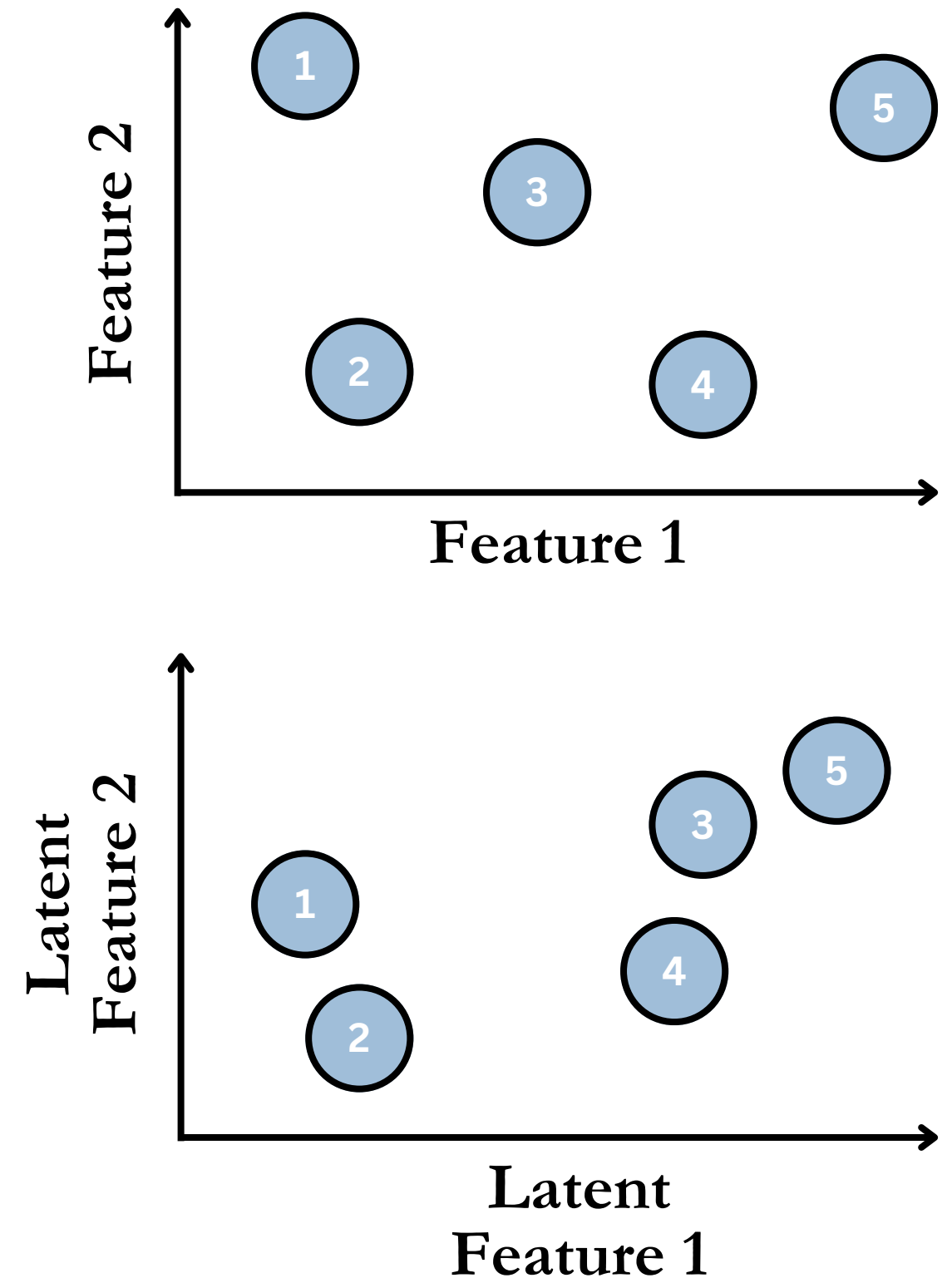
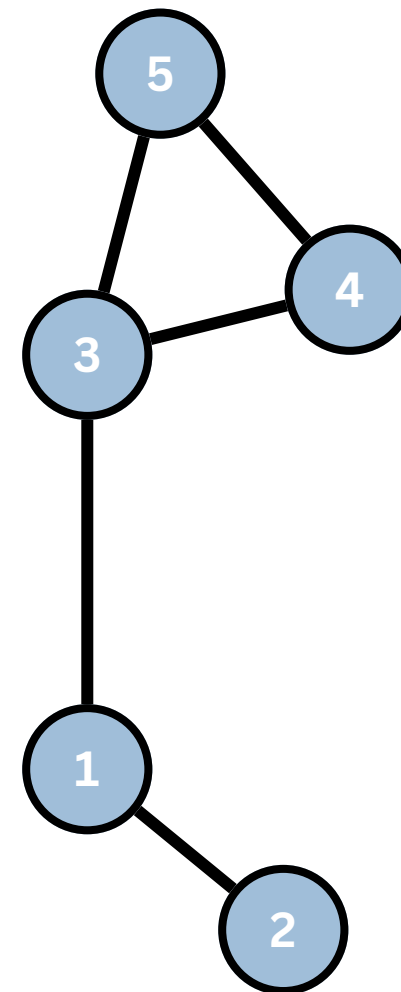
In most applications the number of labelled nodes is quite small compared with the size of the graph. Moreover the training is performed just on a single instance of the graph.

Nodes Embedding

Nodes in a network can be represented in a lower dimensional space

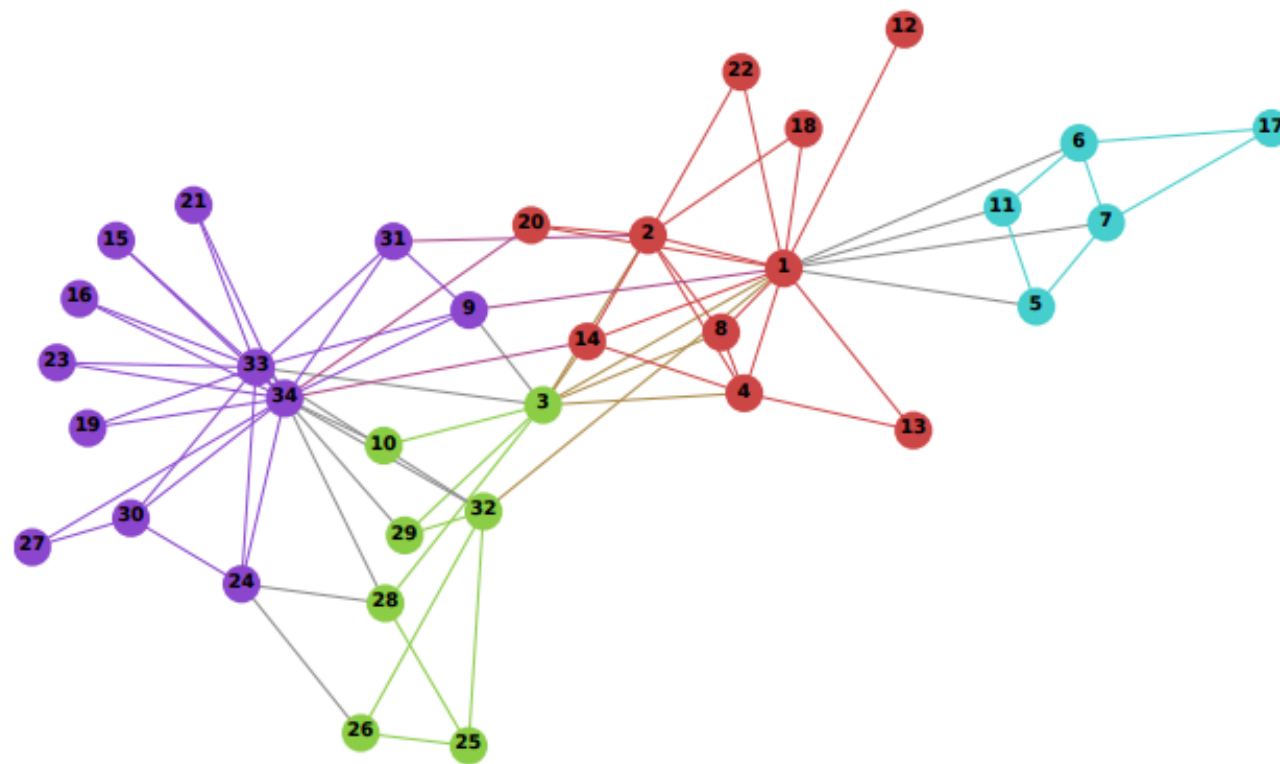
- initially each node is characterized by a vector of features and its position
- we want to combine these features together and with those of other nodes
- the result are new vector (latent features) that describe the nodes better, having also information about neighbors

This is conceptually similar to text embedding

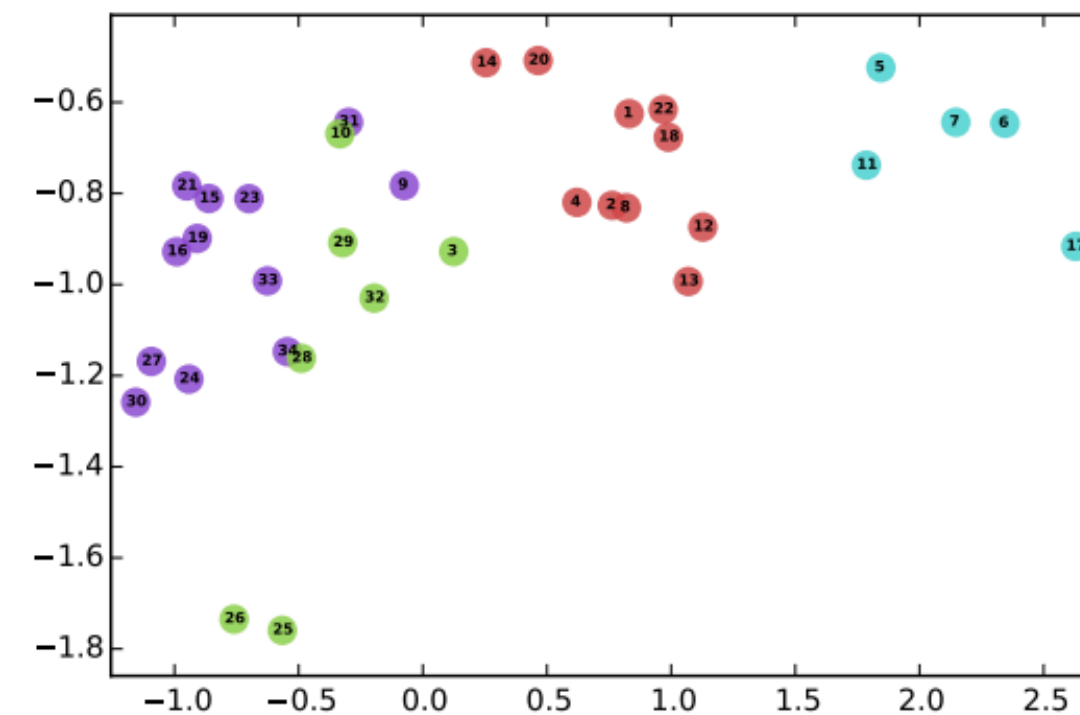


Example: Karate Club

For instance we could get node embedding just looking at the structural aspect. Here the idea is that two nodes will have similar embedding if they are linked in the network or, more generally, if they have high similarity in the network.



(a) Input: Karate Graph



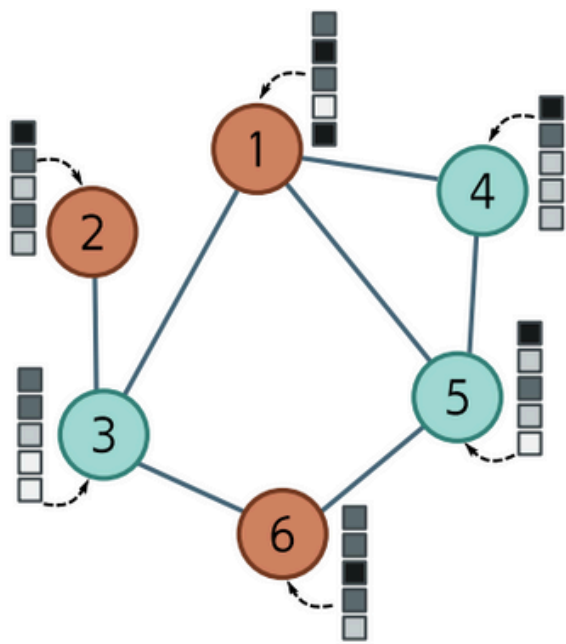
(b) Output: Representation

Using Nodes Embedding

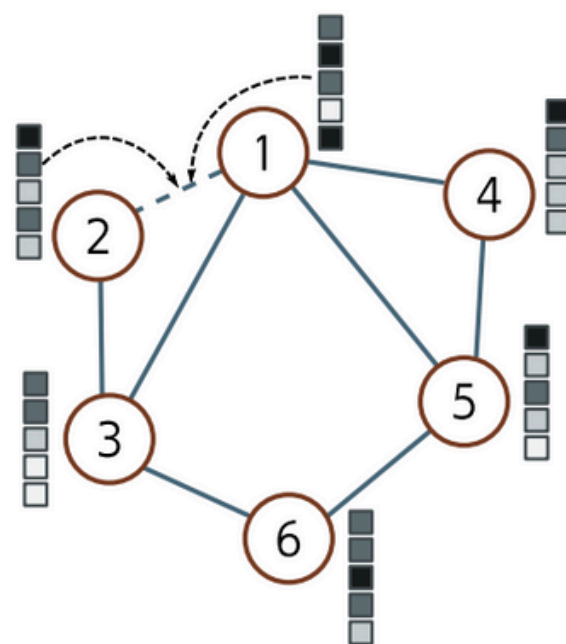
Embedding vectors can be used as input for other machine learning algorithms for several possible tasks

- node classification (is country a tax heaven)
- link prediction (will country export a product)
- graph classification (is molecule poisonous)

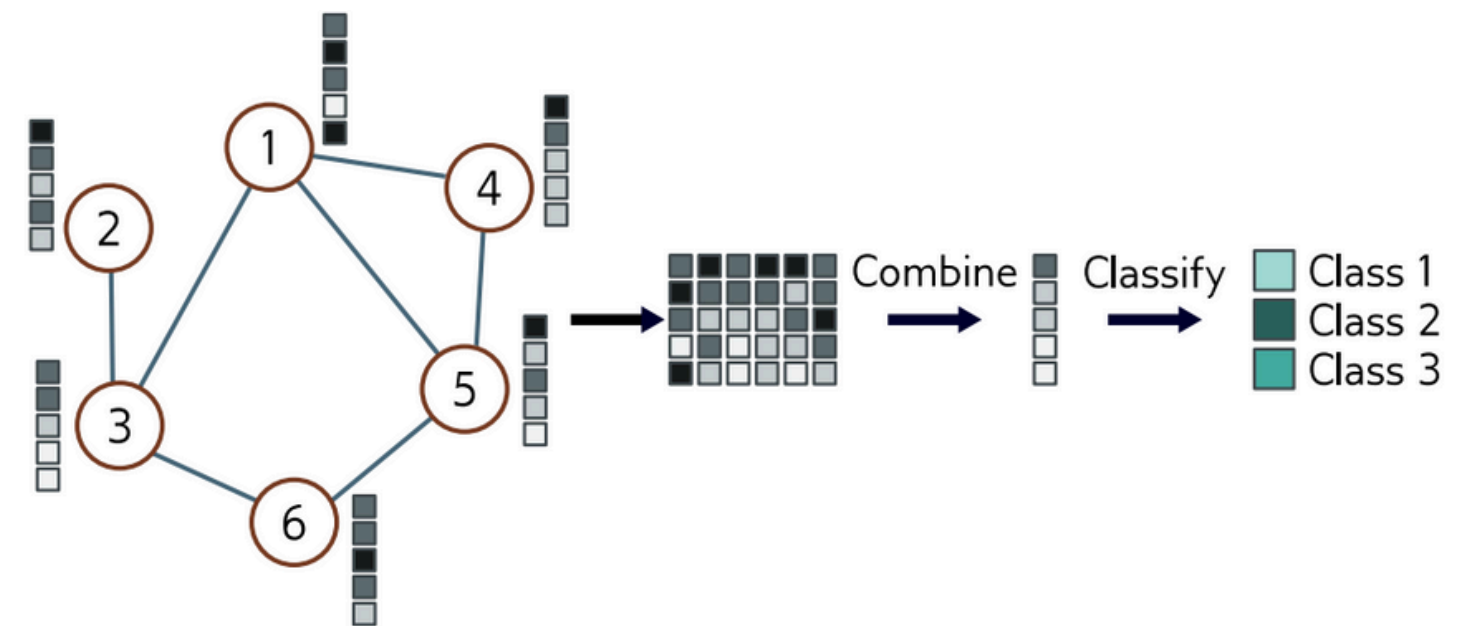
Node Classification



Link Prediction



Graph Classification



A network graph with nodes and edges on a blue background. The nodes are represented by small circles, some of which are black and others are light gray. The edges are thin lines connecting the nodes, forming a complex, interconnected network. The background is a solid light blue color. The title 'Graph Neural Networks' is centered in the image in a large, bold, white font.

Graph Neural Networks

Convolutional Neural Networks

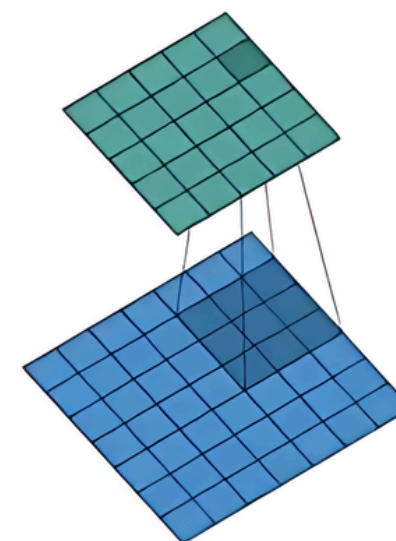
Images can be seen as special graphs:

- each pixel is a node
- each node is connected to the 8 closest pixels
- each node has a feature vector (B/W 1 number, color 3 numbers)

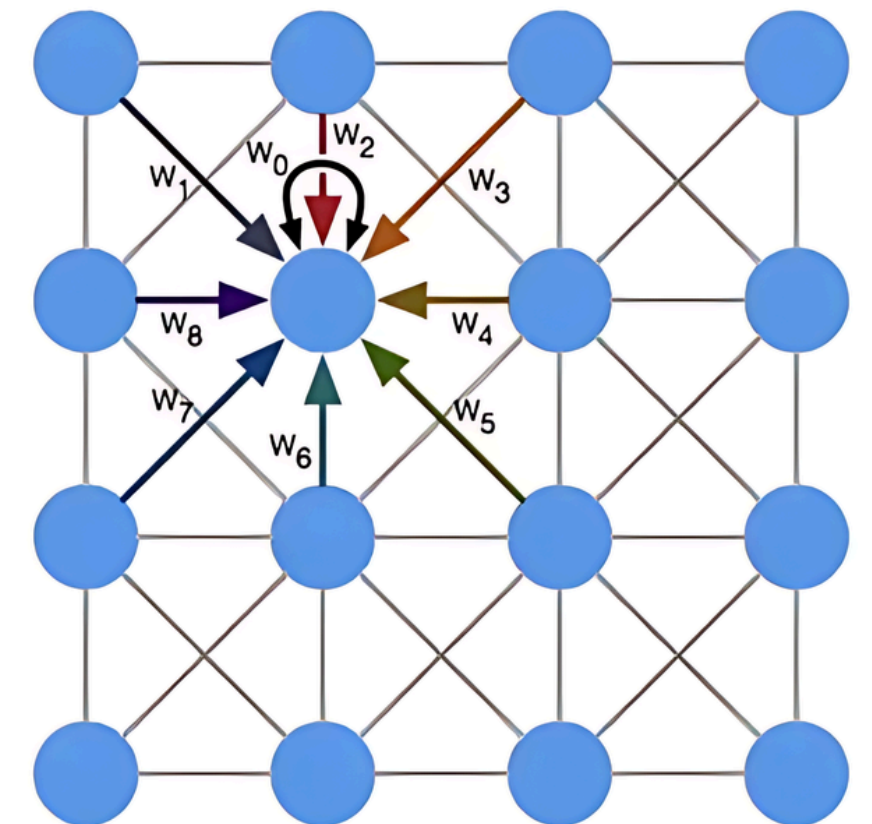
The convolution is combining all the information coming from a pixel and its neighbors \mathbf{x} into a single value h_i

$$h_i = a \left(W_0 x_i + \sum_{n=1}^8 W_n x_n \right)$$

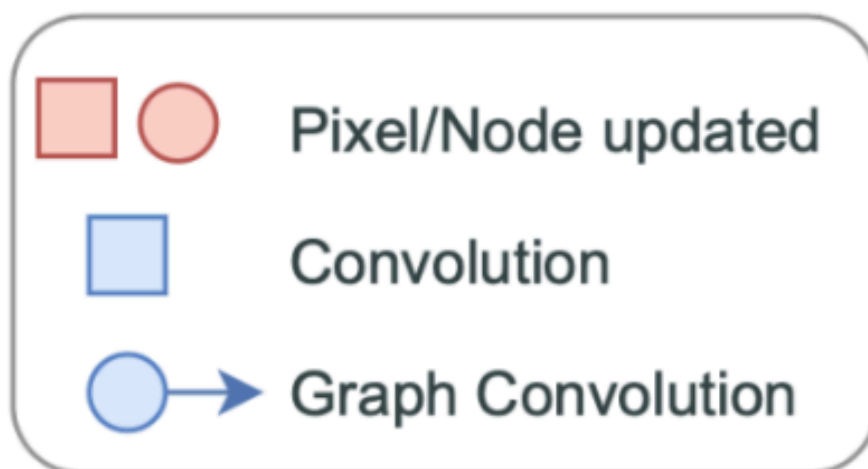
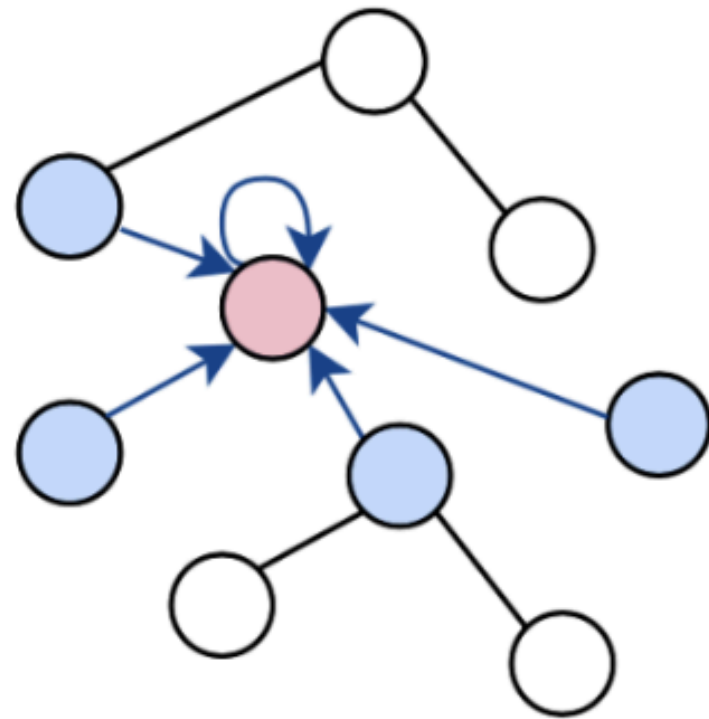
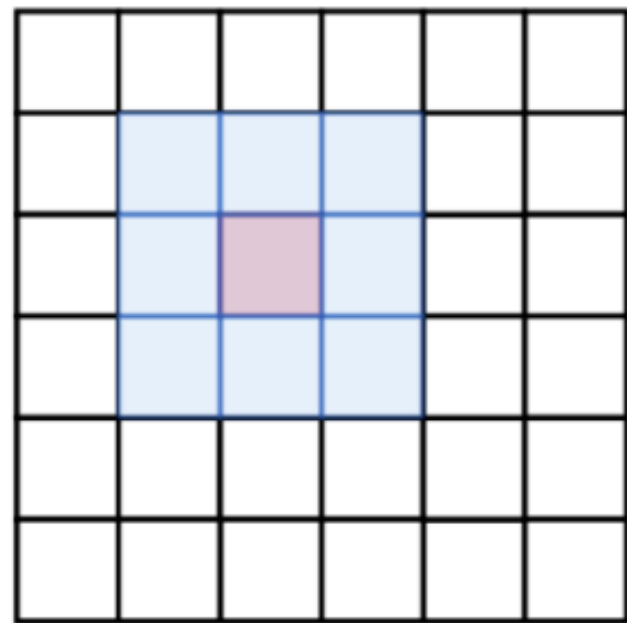
Latent
Representation \mathbf{h}



Input
Image \mathbf{x}



From Images to Graphs



We want to generalize the procedure to arbitrary graphs:

- now the number of neighbors is no longer fixed
- we can not use kernels of fixed size to learn parameters

Despite these limits the idea is the same, we want to use the graph structure to combine features coming from neighbors. Each node will have different number of incoming contributions.

Translating this into Math

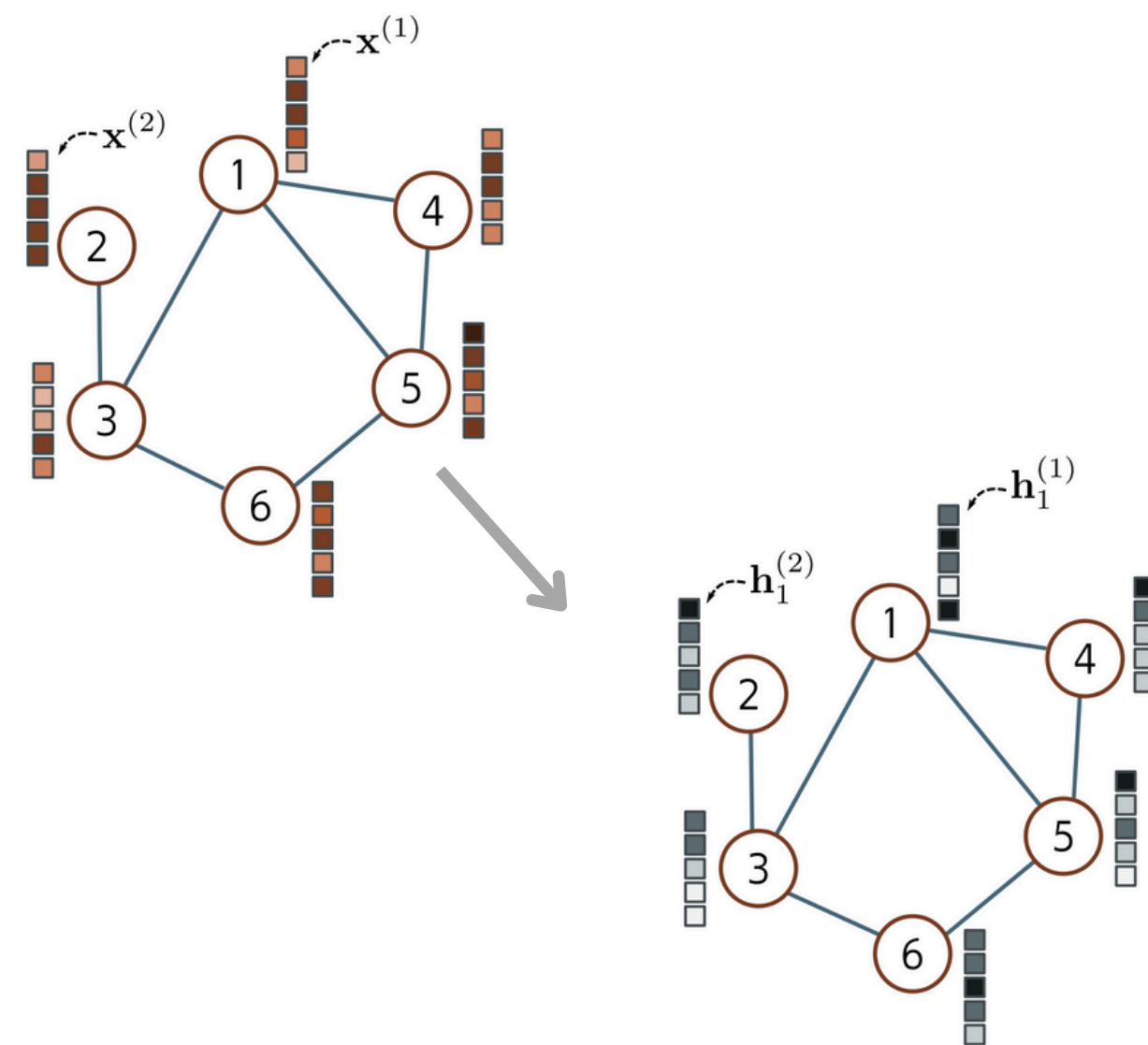
Let us for one moment forget about the trainable weights. If we want to combine information from close neighbors we can use the adjacency matrix

$$\mathbf{h}_i = a \left(\mathbf{x}_i + \sum_j^N A_{ij} \mathbf{x}_j + \beta_i \right)$$

Note that we add to explicitly include the self-loop, otherwise we lose information from the node itself.

We set $\bar{A} = A + I$

$$\mathbf{h}_i = a \left(\sum_j^N \tilde{A}_{ij} \mathbf{x}_j + \beta_i \right)$$



Adding Learnable Shared Weights

What type of parameters do we want to learn?

- in CNN the weights are inside the filters. These weights are weighting in a different way all the contributions coming from the 8 different neighbors
- this approach is not feasible in our case since there is no fixed number of neighbors
- instead we want the weights to learn relations among features
- these relations are independent on where the node is placed on the graph

This is achieved by multiplying the input features by a matrix of learnable parameters \mathbf{W}

$$\mathbf{h}_i = a \left(\sum_j^N \tilde{A}_{ij} \mathbf{W} \cdot \mathbf{x}_j + \beta_i \right)$$

The size of \mathbf{W} is $H \times D$, where D is the input dimension and H the latent features dimension

What are the Weights Doing?

Let's try to better understand what the weights are doing

- we assume each node to have two features
 - \mathbf{x}_j is a vector with two components
- we set the latent dimension H to 3
 - \mathbf{W} is a 3x2 matrix

When we multiply the matrix with the feature vector, we obtain a new feature vectors with 3 components that contains linear combinations of the original vectors. This is analogous to what we would do in a perceptron

$$\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \cdot \begin{pmatrix} x_{j1} \\ x_{j2} \end{pmatrix} = \begin{pmatrix} w_{11}x_{j1} + w_{12}x_{j2} \\ w_{21}x_{j1} + w_{22}x_{j2} \\ w_{31}x_{j1} + w_{32}x_{j2} \end{pmatrix}$$

The Graph Convolutional Layer

We are almost there

- the layer we defined in the previous slides has a small issue
- every time we apply the operation we are summing many vectors, one for each neighbor
- if we built a Deep Neural Network, layer after layer these values will grow causing problems

In order to solve this issue we need to add a normalization. The standard choice is

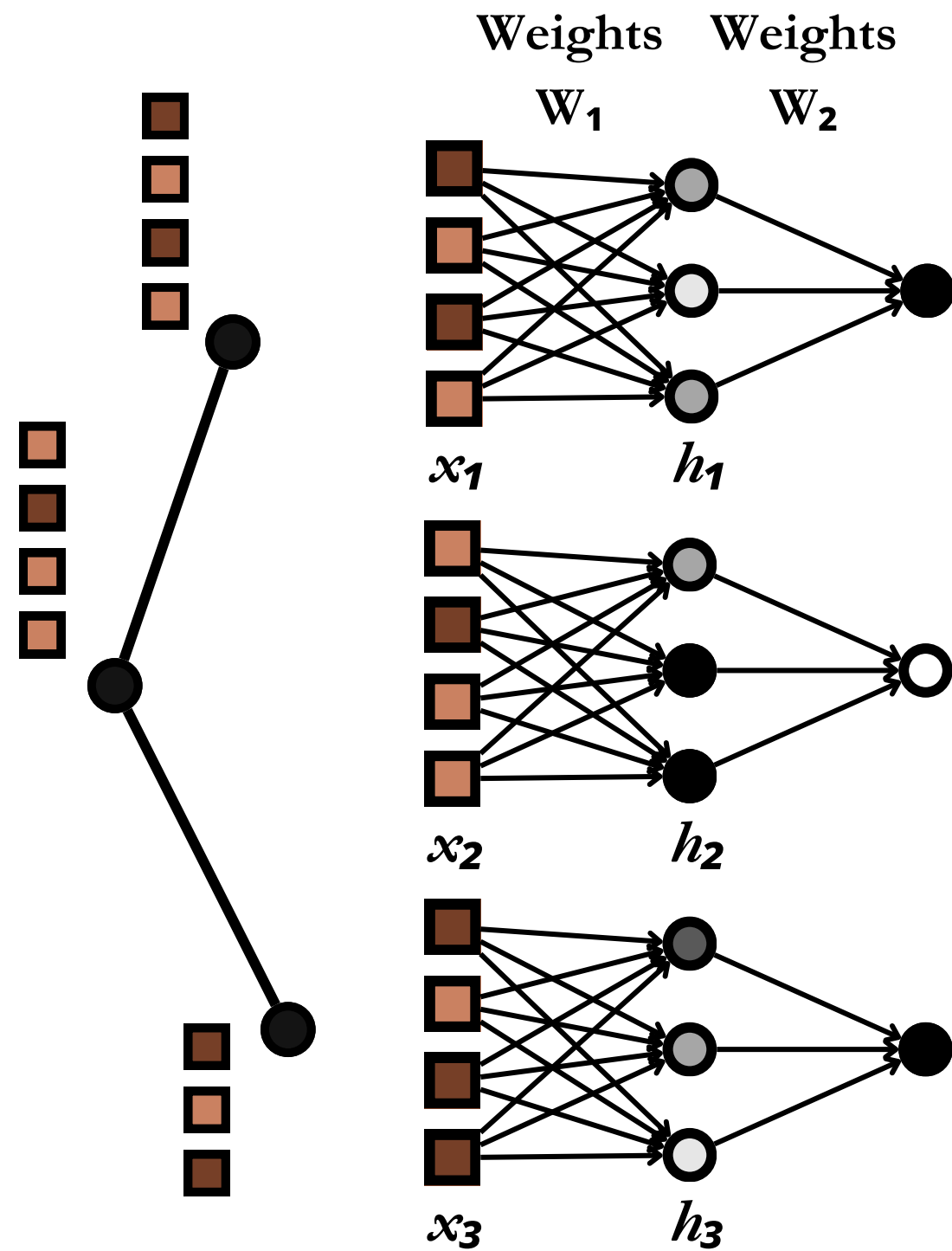
$$\mathbf{h}_i = a \left(\sum_j^N \frac{\tilde{A}_{ij}}{\sqrt{d_i d_j}} \mathbf{W} \cdot \mathbf{x}_j + \beta_i \right)$$

Here d_i denotes the degree of node i (number of connections). This expression defines the so called Graph Convolutional Layer. Similarly to a CNN we are aggregating spatial information and we are reusing parameters in different locations of the graph.

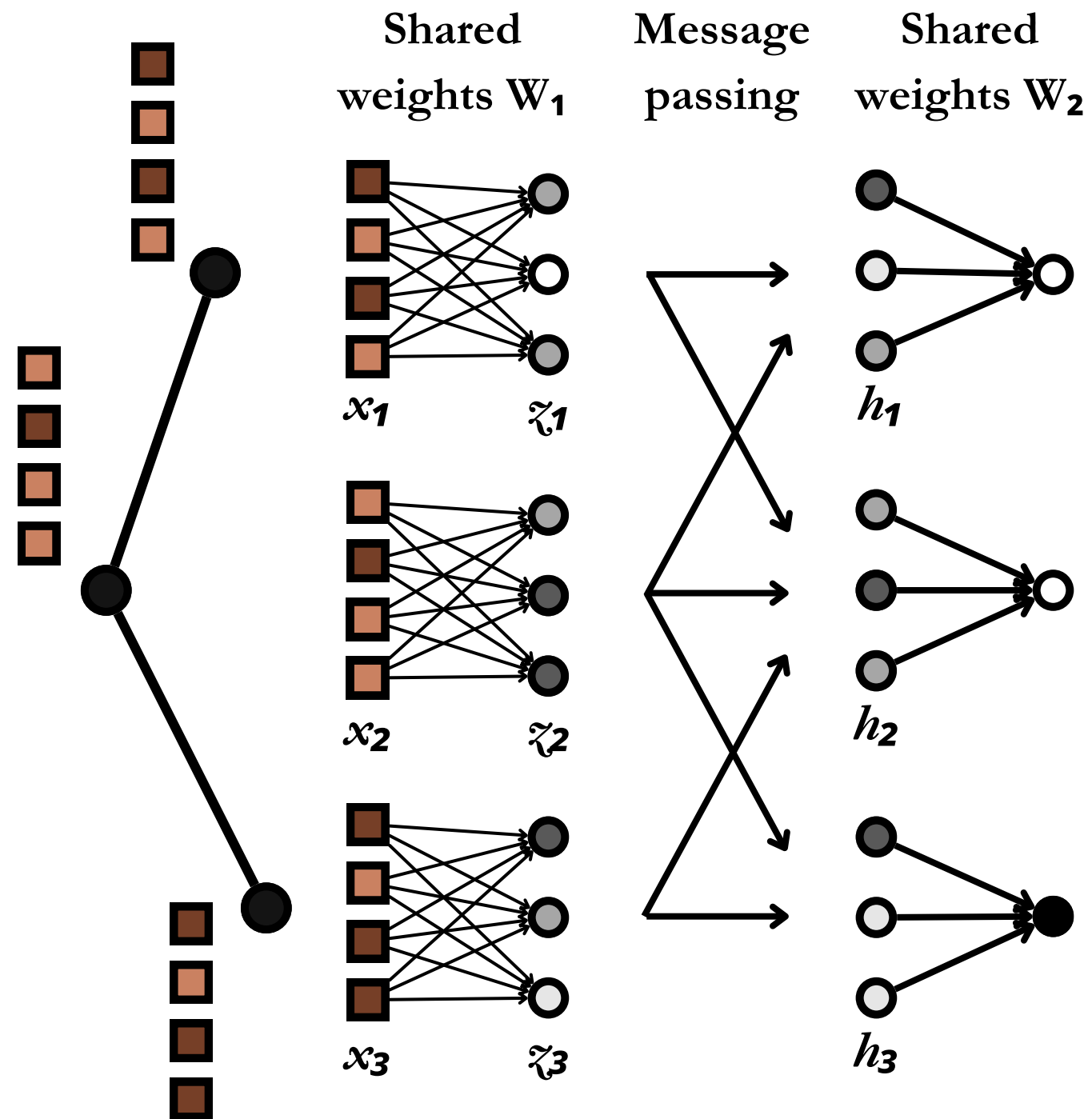
MLP Approach

First we consider the problem from a MLP perspective

- each node i is characterized by a vector of features \mathbf{x}_i
- we focus on these vectors and we completely discard the graph
- we train a MLP using all the vectors in the training set and then we use it in the test set
- the hidden layers of the MLP produce latent representation of the data \mathbf{h}_i
- this is the standard procedure for MLP classification



GNN Approach



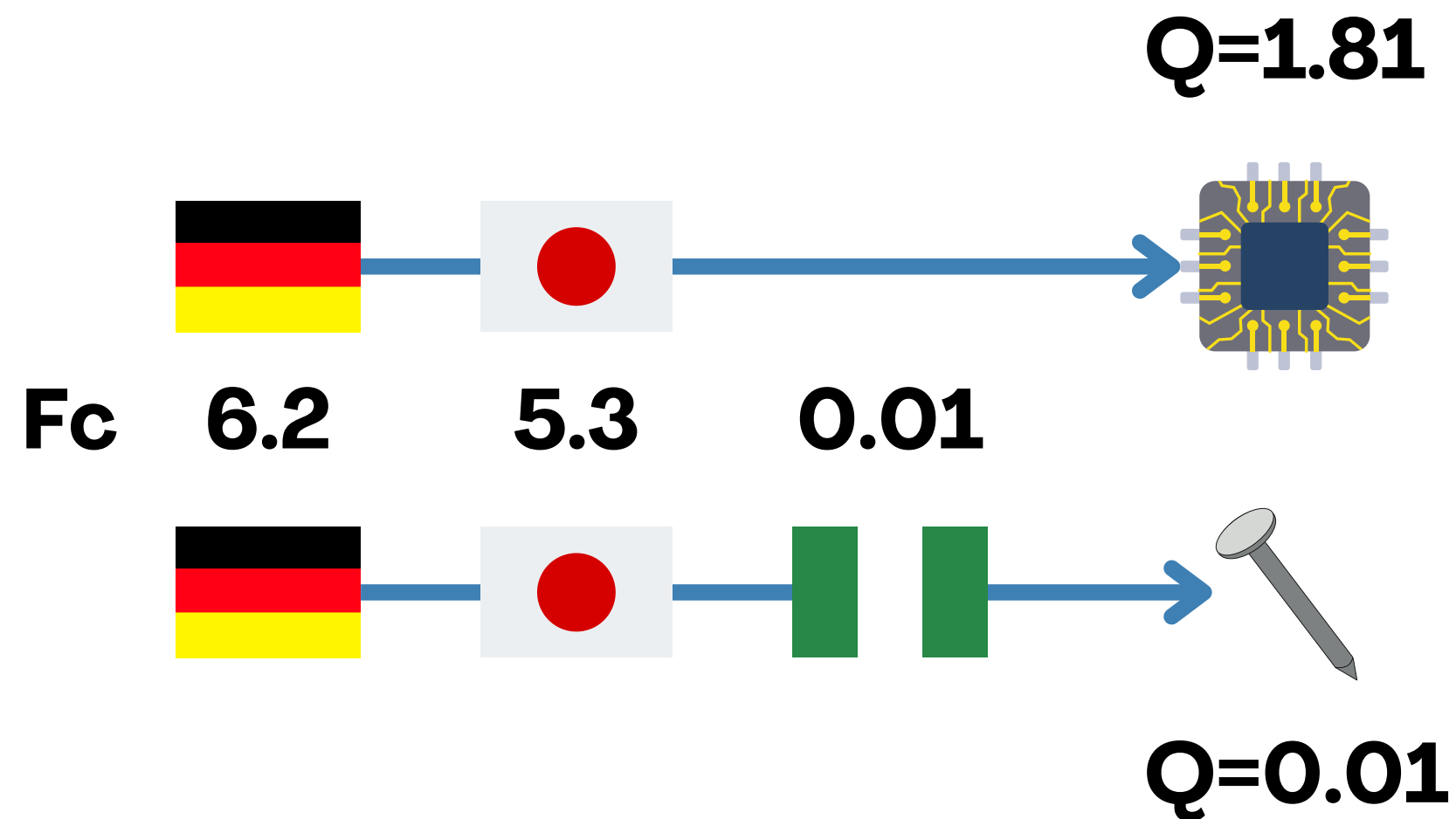
The GNN works in a similar way, but there is sharing of the latent representation

- each node i is characterized by a vector of features \mathbf{x}_i
- we focus on these vectors and also on the graph structure
- now a dense layer is used to create an intermediate representation of features \mathbf{z}_i
- these representations are combined using the graph structure to get the latent representation \mathbf{h}_i
- these latent representations are then used for the classification

A network graph visualization on a blue background. The graph consists of numerous nodes (represented by small circles) and edges (represented by thin lines). The nodes are arranged in a complex, interconnected pattern, with some nodes having a higher degree of connectivity than others. The edges are thin and light blue, while the nodes are small circles in various shades of blue and white. The overall structure is dense and interconnected, with some clusters and some isolated nodes.

Fitness Centrality

Economic Fitness and Complexity



Economic Fitness and Complexity was originally introduced to work with bipartite networks

- it quantifies fitness of countries and complexity of products
- is a non-linear algorithm

$$F_c^{(n)} = \sum_p M_{cp} Q_p^{(n-1)}$$

$$Q_p^{(n)} = \frac{1}{\sum_c M_{cp} \frac{1}{F_c^{(n-1)}}}$$

The idea is that the countries giving more information are those only exporting few, low complexity products

Cristelli, M., Gabrielli, A., Tacchella, A., Caldarelli, G., & Pietronero, L. (2013). *Measuring the intangibles: A metrics for the economic complexity of countries and products*. PloS one, 8(8), e70726.

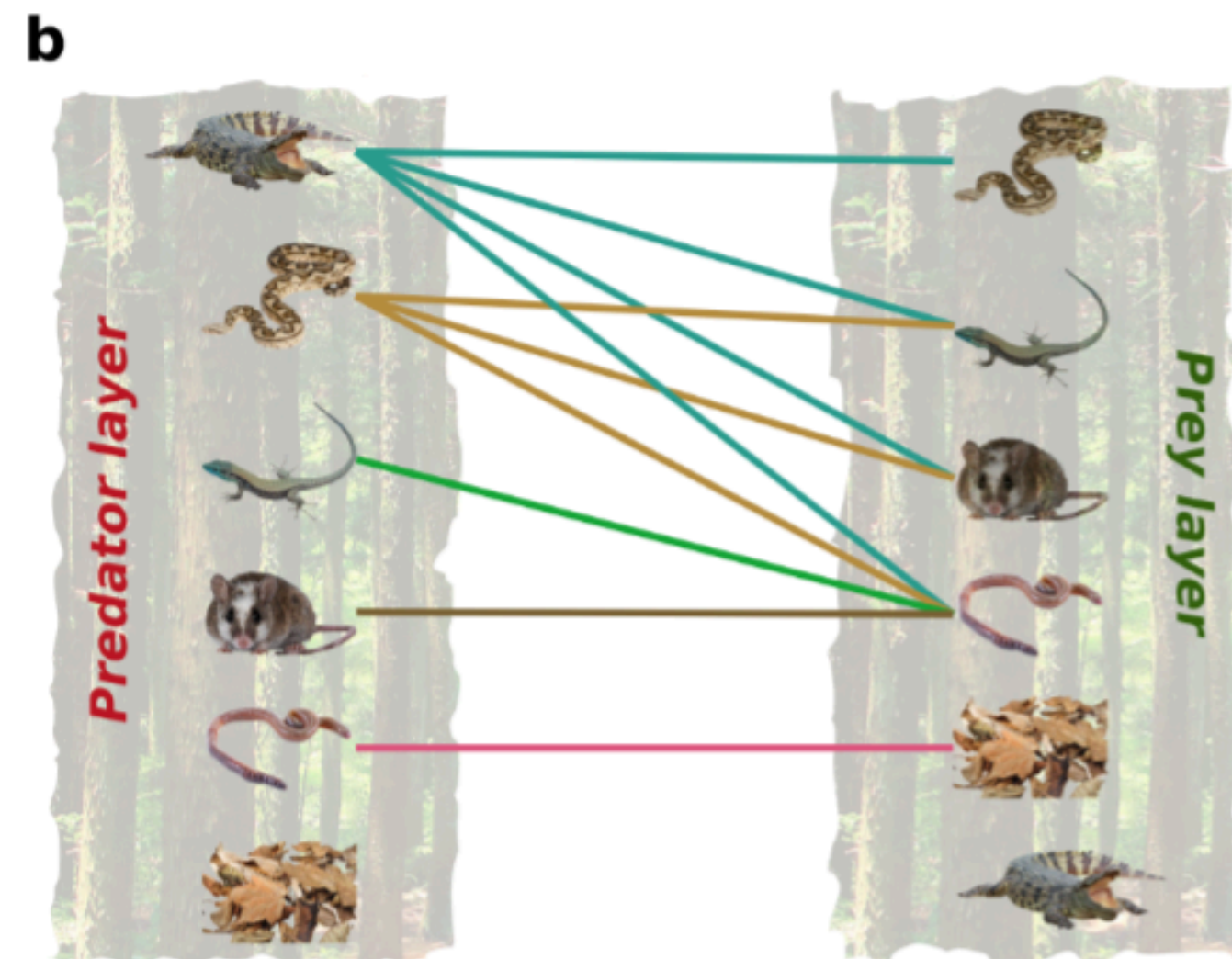
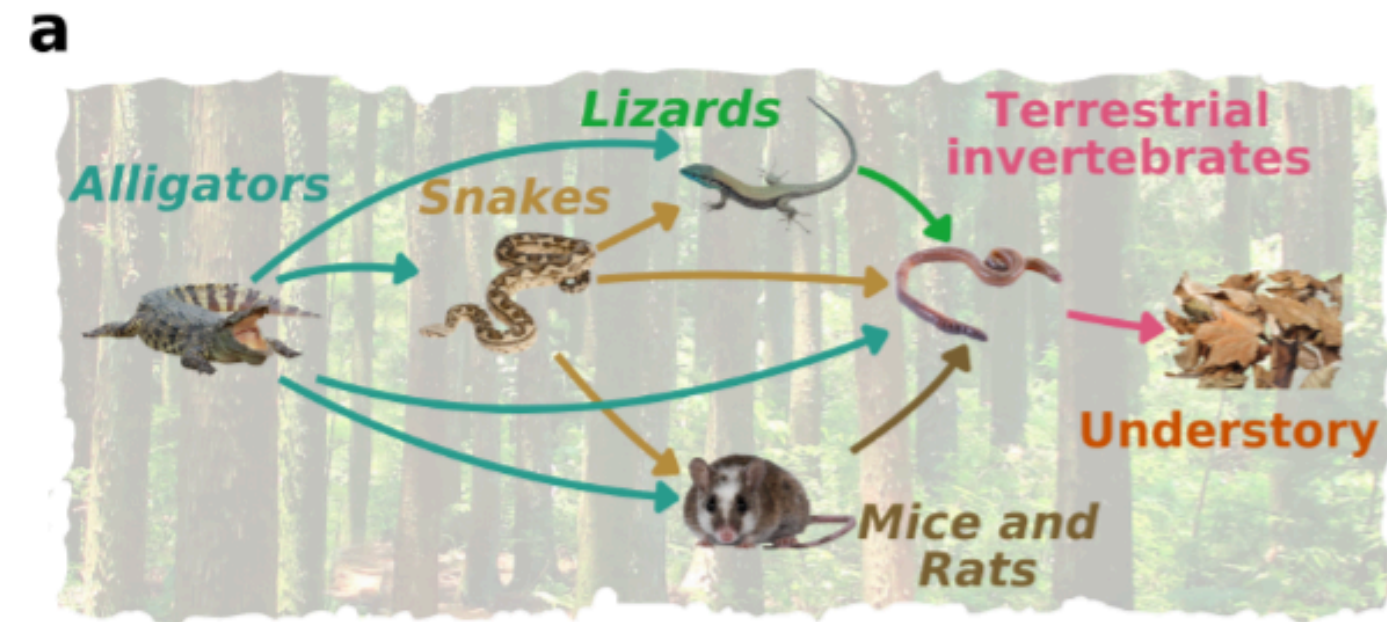
Generalization to Directed Networks

The EFC algorithm can be generalized to arbitrary networks

- we start considering a food web (directed network)
- each species has a double nature, both as prey and predator
- in the original food web we have arrows going from the predator to the prey

We can represent the same network as a bipartite networks with two layers

- predator layer
- prey layer



Fitness and Importance of Species

Now each species will be characterized by two quantities

- **Importance I**

- Measures how essential a species is as prey
- High importance if preyed upon by many low-fitness species
- Critical for food web stability
- Inverse of Complexity

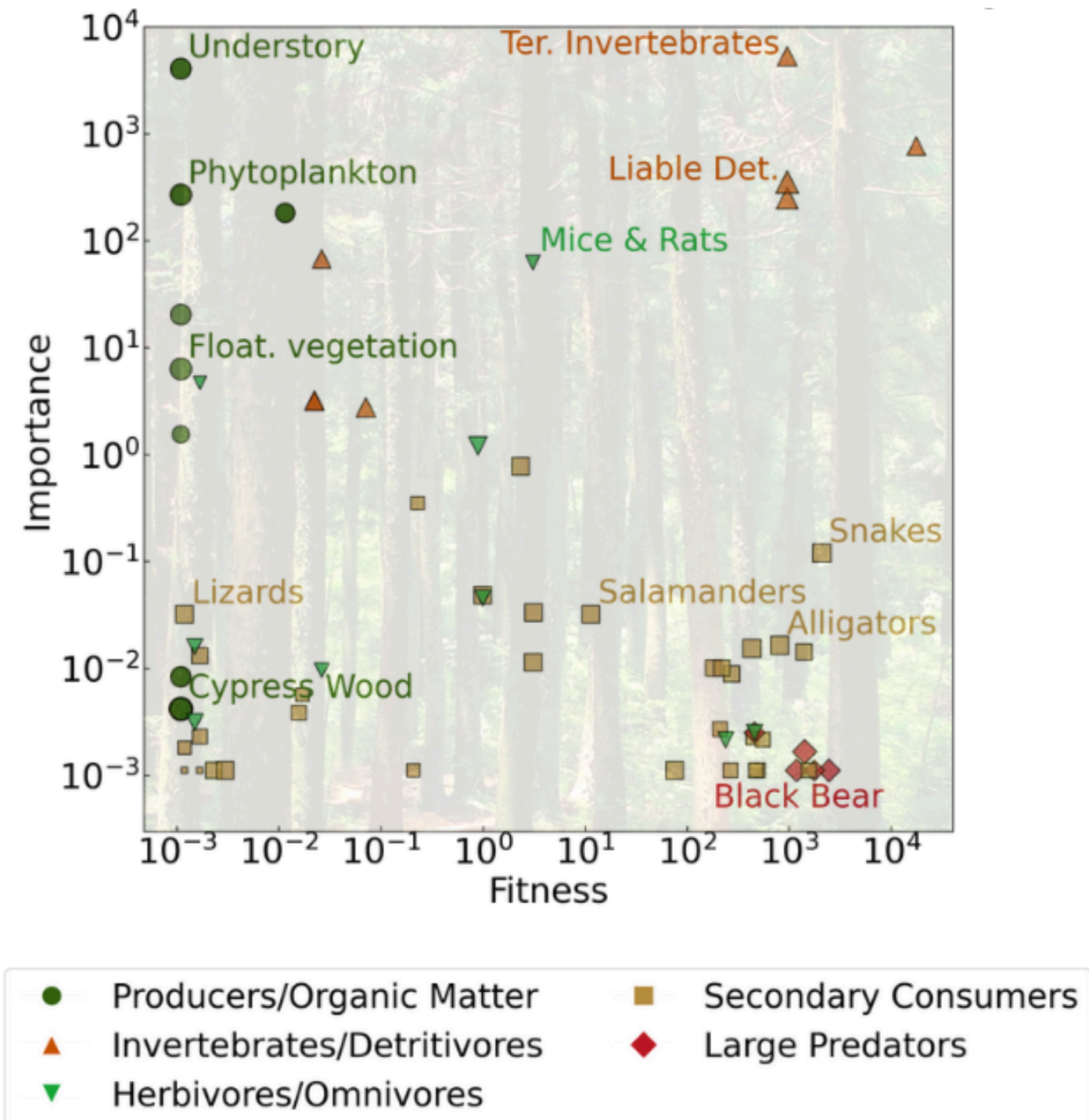
- **Fitness F**

- Measures predatory capabilities
- High fitness if species preys on diverse, low-importance (high-compl.) species
- Reflects adaptability in food web

We have the same algorithm as EFC, but expressed in terms of importance

$$\begin{cases} F_i^{(n+1)} = \delta + \sum_j M_{ji} / I_j^{(n)} \\ I_i^{(n+1)} = \delta + \sum_j M_{ij} / F_j^{(n)} \end{cases} \quad F_i^{(0)} = I_i^{(0)} = 1 \quad \forall i$$

Fitness-Importance Plane



Each species is characterized by two quantities

- we focus on the Cypress dry season food web
- we can place each species on the fitness-importance plane

We observe regular patterns. For instance

- large predators are in the bottom right corner
- producers are in the top left corner

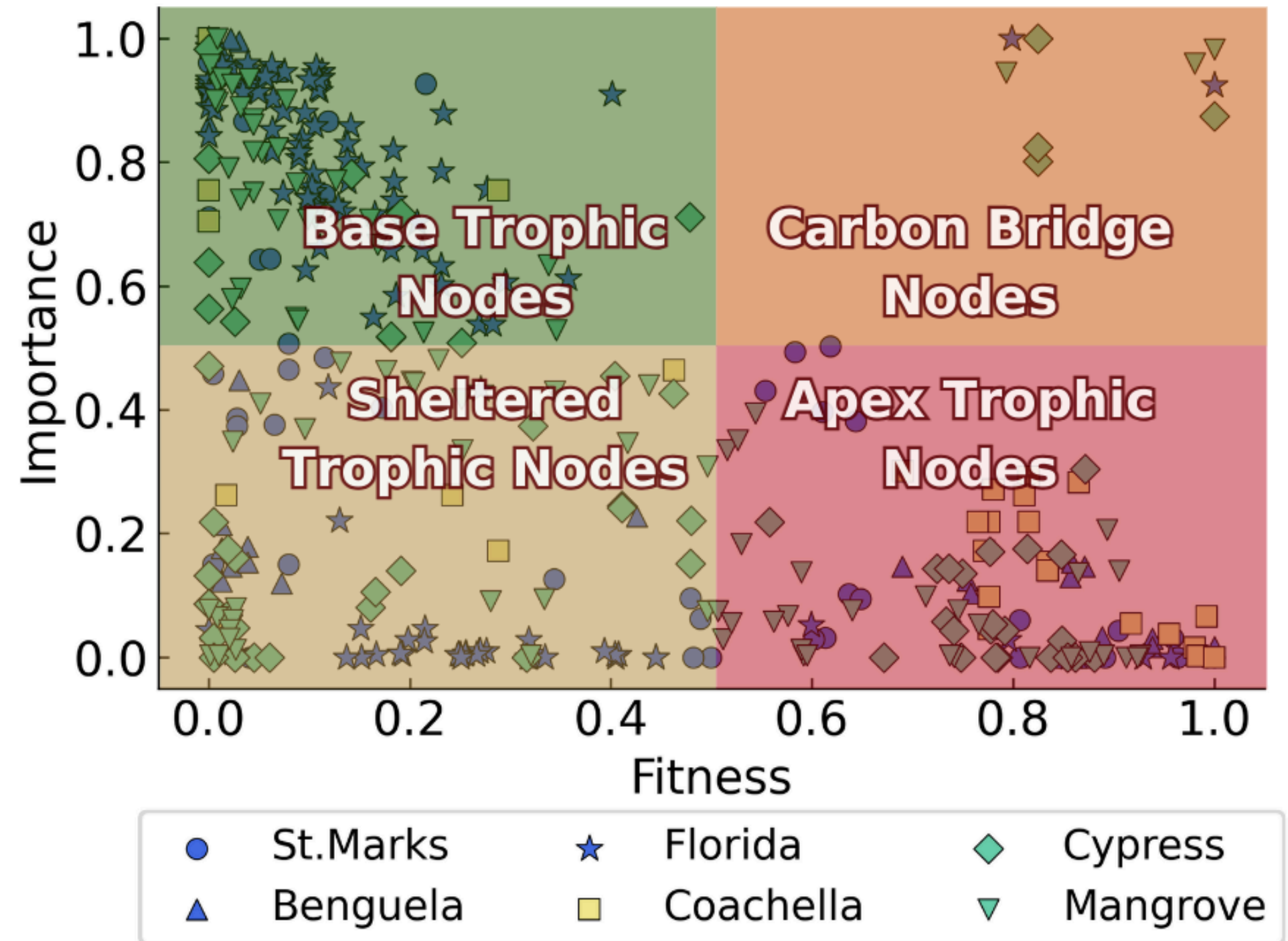
The Four Quadrants

We can repeat the same analysis for several ecosystems

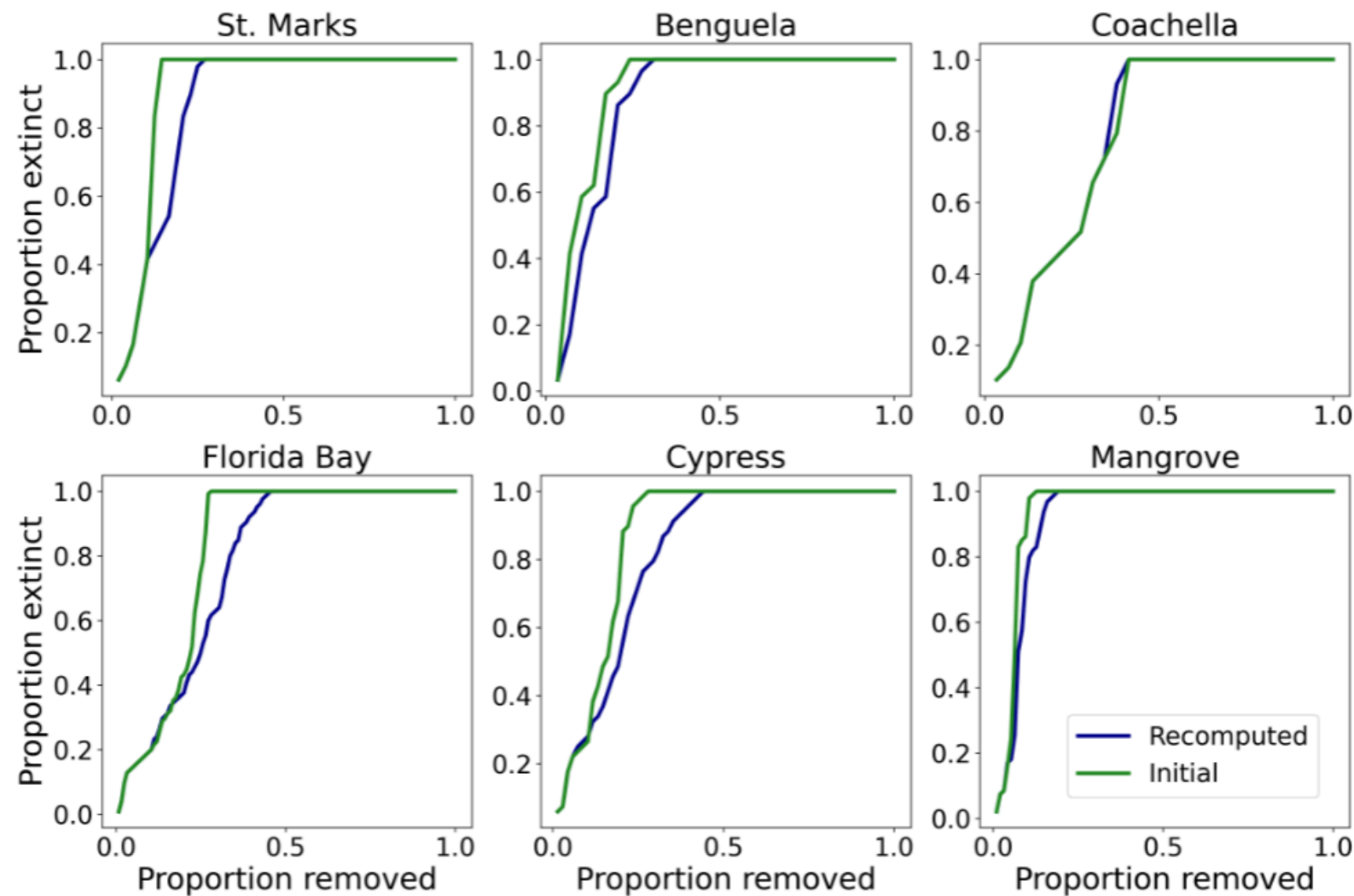
- we observe a similar disposition of species

We can divide the fitness–importance plane in 4 regions

- **Basic trophic nodes**
 - High I, low F
- **Carbon bridge nodes**
 - High I, high F
- **Sheltered trophic nodes**
 - Low I, low F
- **Apex trophic nodes**
 - Low I, high F



Extinction Curves



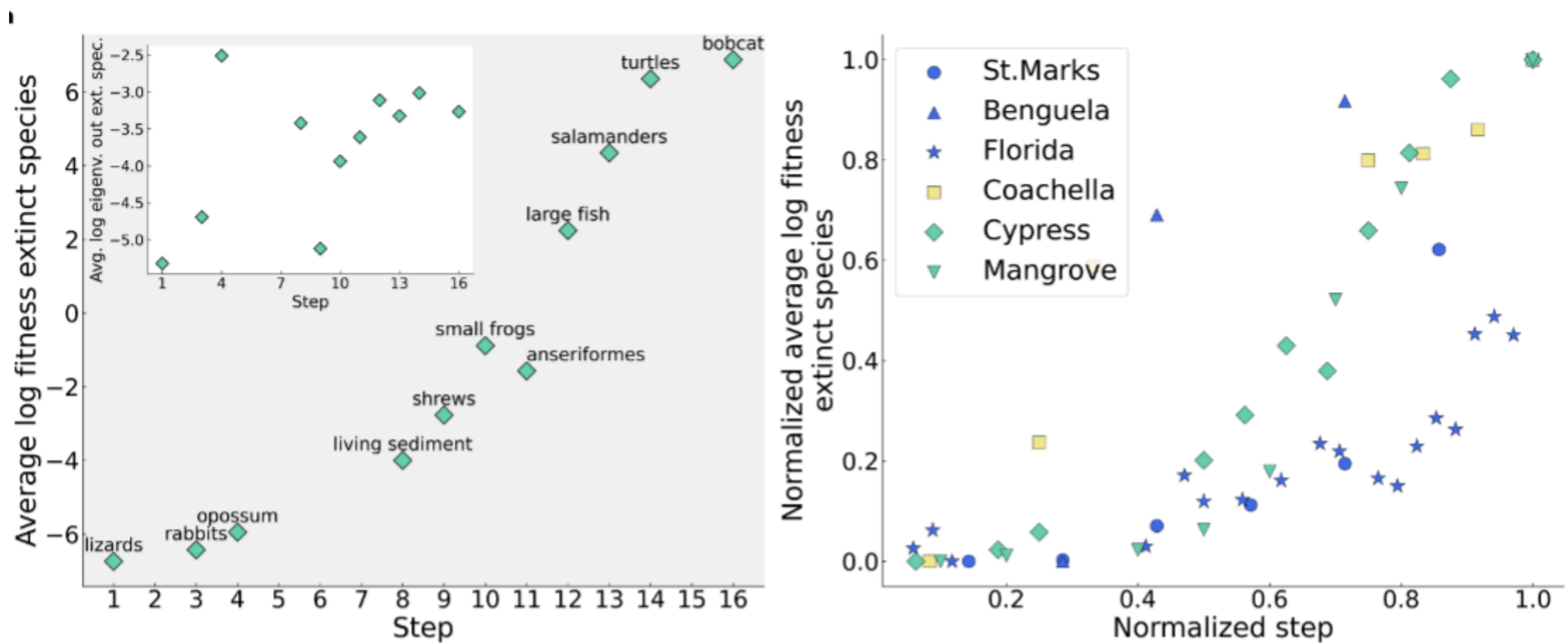
High importance nodes are expected to trigger extinction cascades

- we can look at the usual error/attack curves
- we remove species following their importance
- all nodes that get disconnected become extinct
- the larger the area under the curve, the more species get rapidly extincted

Importance performs similarly with respect to eigenvector centrality

Fitness and Vulnerability

Low fitness species are also the most vulnerable, since they rely on just few species for food. Low fitness species are those that become extinct earlier during the extinction cascades



Undirected Graphs

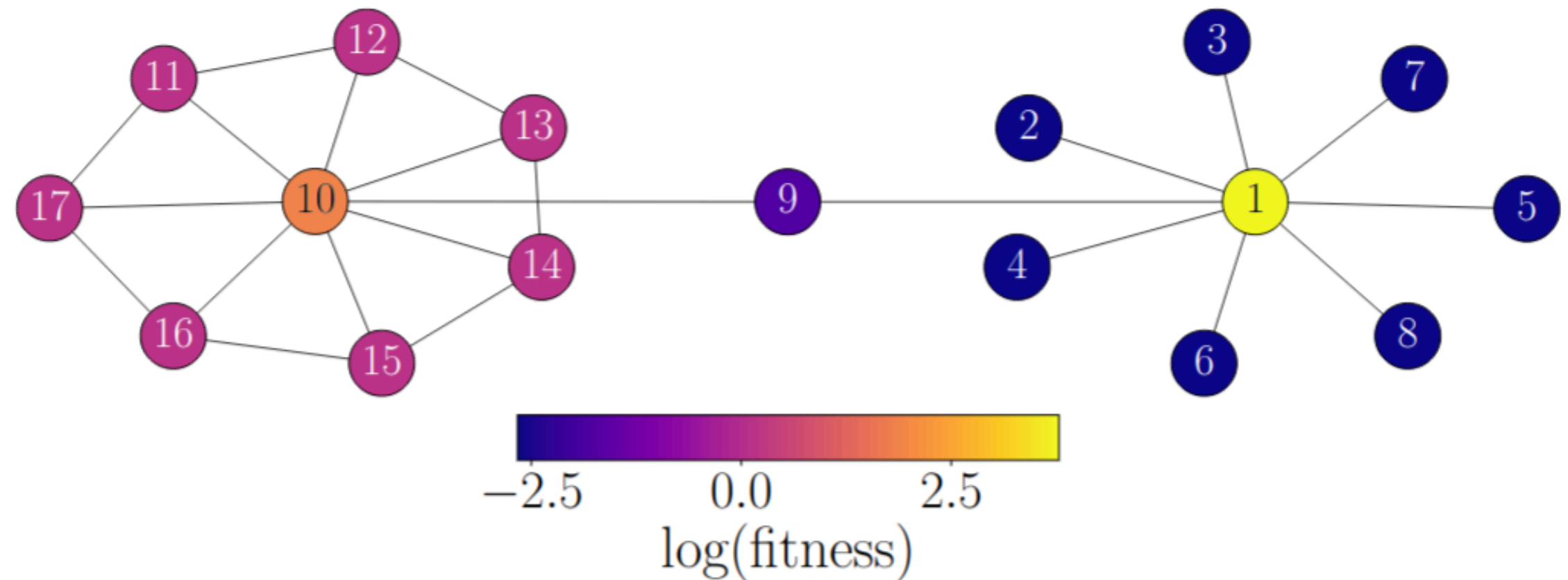
We can generalize the idea of fitness also to undirected graphs

- in this case each node will only have a single quantity associated
- we call it fitness centrality

A node has an high fitness centrality if it is connected to many nodes with low fitness centrality.

Mathematically we have the equation

$$F_i^{(n+1)} = \delta + \sum_{j=1}^N A_{ij} \frac{1}{F_j^{(n)}}$$



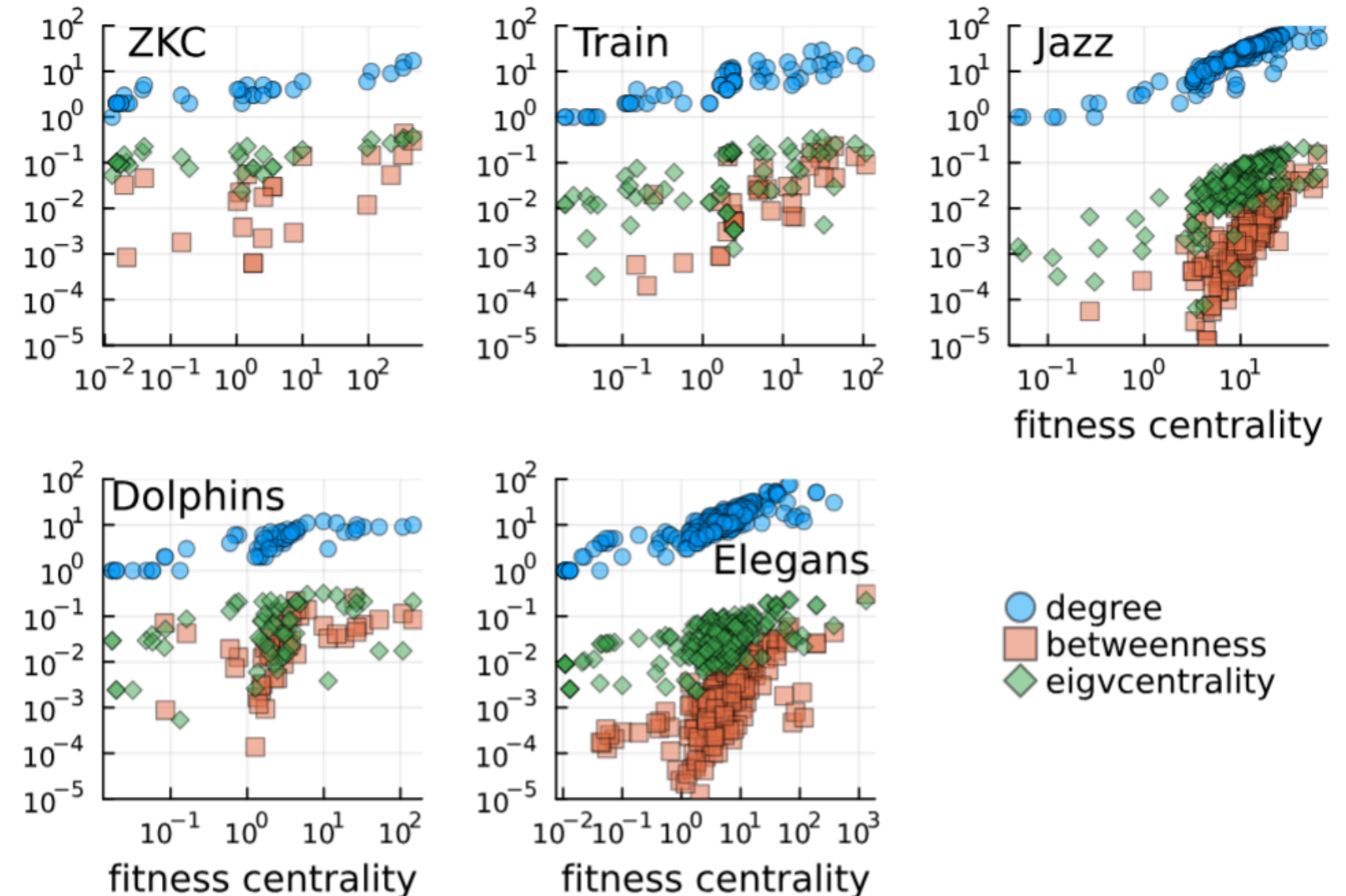
Servedio, Vito DP, et al. "Fitness Centrality: a non-linear centrality measure for complex networks." Journal of Physics: Complexity (2025).

Fitness vs Other Measures

We need to understand if the fitness can detect information not provided by other measures

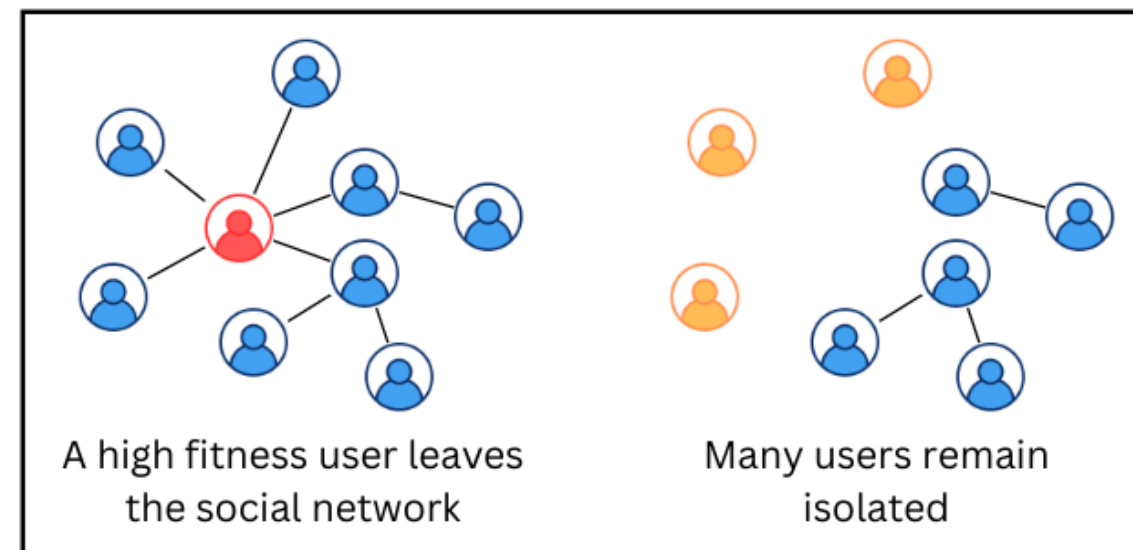
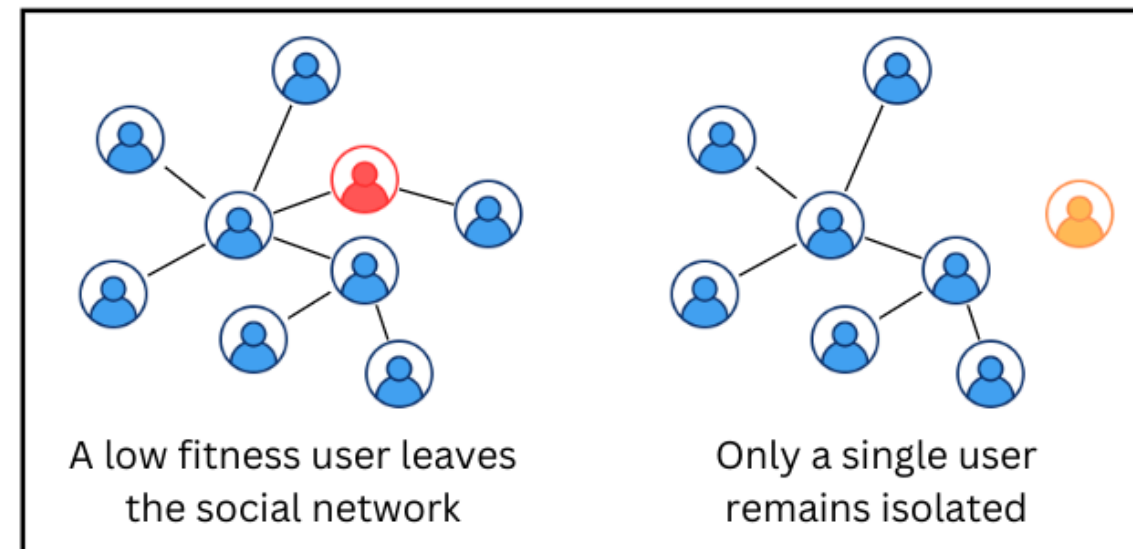
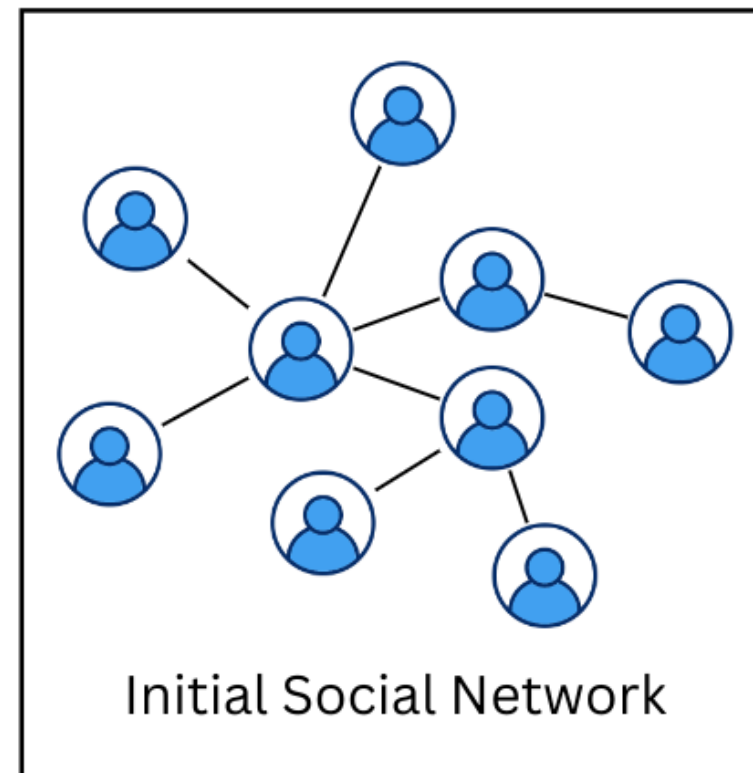
- Fitness centrality is correlated with the degree
- correlations with other standard measures is low
- The information captured by fitness centrality is different

The differences derive from the non-linear nature of the algorithm



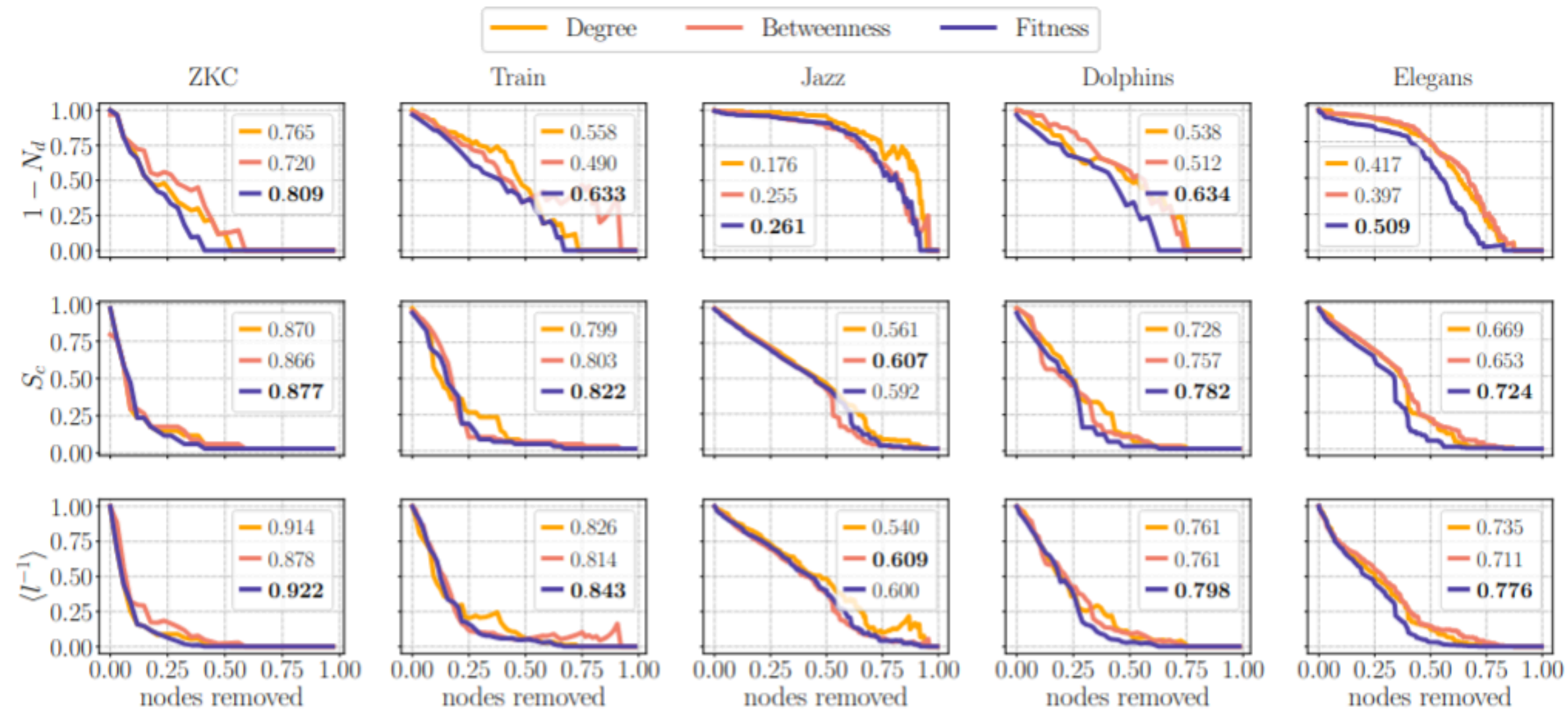
Detecting Crucial Users

As in the case of the food web, fitness centrality can be used to understand which users, if removed, would lead to the largest number of isolated nodes



Analysis of Real Networks

Applying fitness centrality to real networks and performing node removal we actually find that fitness centrality outperforms other measures in networks disruption. This is particularly true when the removal sequence must be computed before the attack



Conclusions

Machine Learning on Graphs

Many datasets are structured as graphs, with features associated to nodes and edges. Machine learning on graphs allow to analyze these data, performing operations like node classification or embedding.

Graph Neural Networks

Graph Neural Networks are special neural networks that can work with graph data. They are based on the concept of message passing between neighbors and can be used for all the machine learning tasks.

Fitness Centrality

The economic fitness and complexity algorithm can be generalized to arbitrary graphs. In this context, it allows to detect the most crucial and the most vulnerable nodes within the system.