

# Introduction to graph machine learning

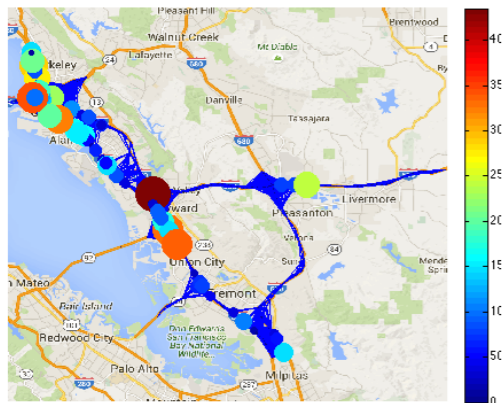
---

Dr Dorina Thanou

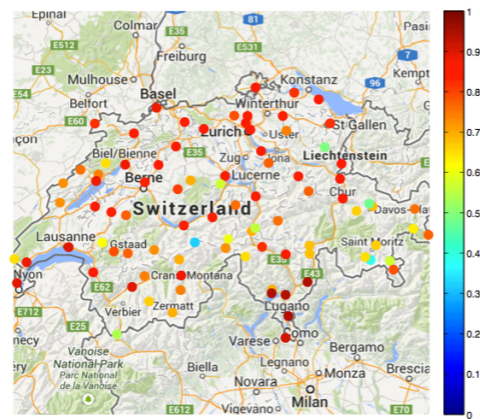
October 20, 2022

# Going beyond graph structure

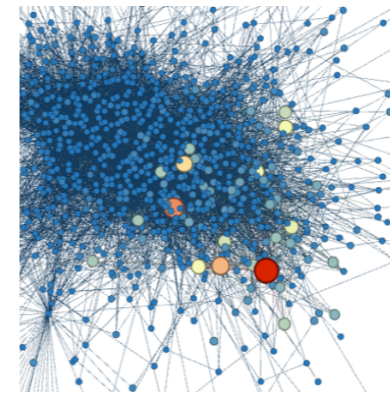
- Very often data comes with additional features
  - Not only graphs, but attributes on the nodes of the graph



Transportation networks



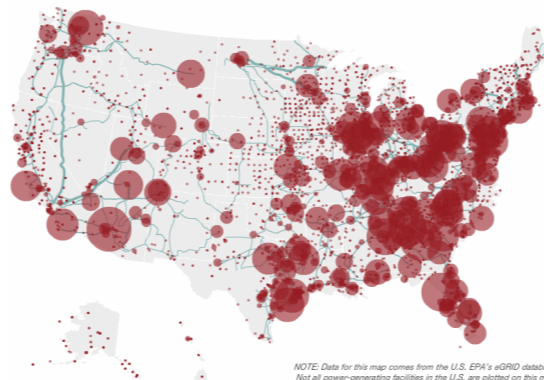
Weather networks



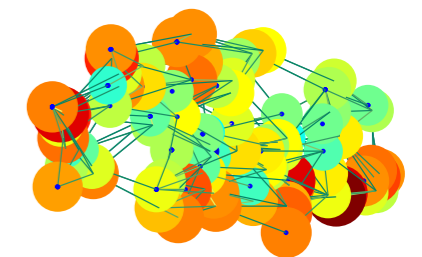
Social networks



Disease spreading networks



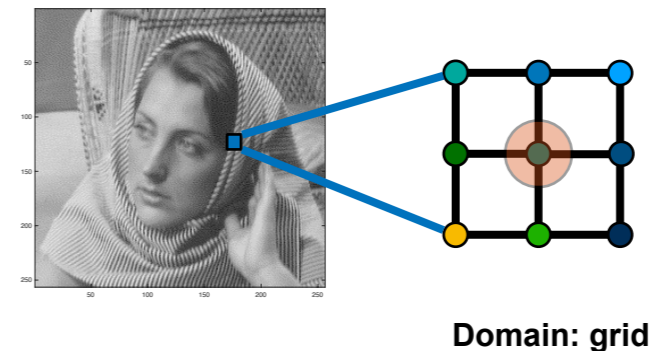
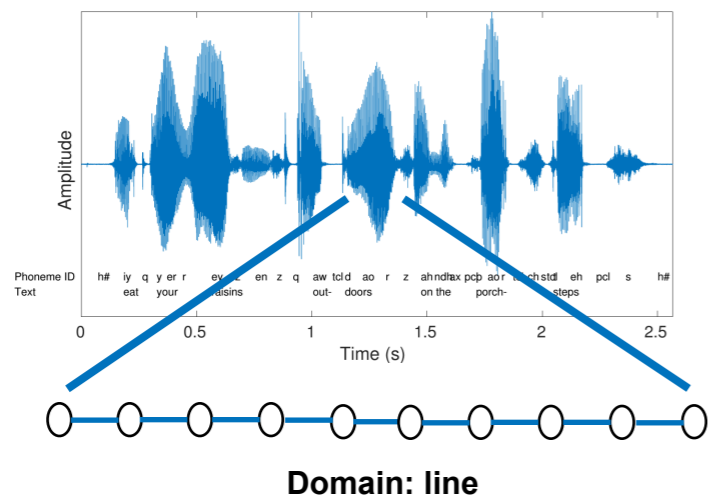
Electric grid networks



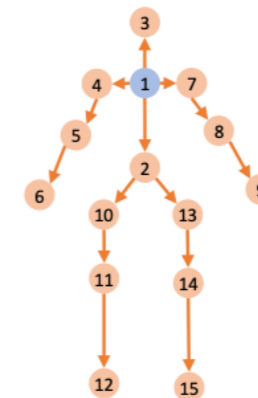
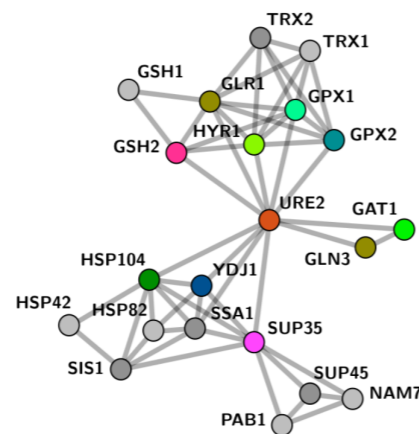
Biological networks

# Graph structured data

- Data live on a regular domain

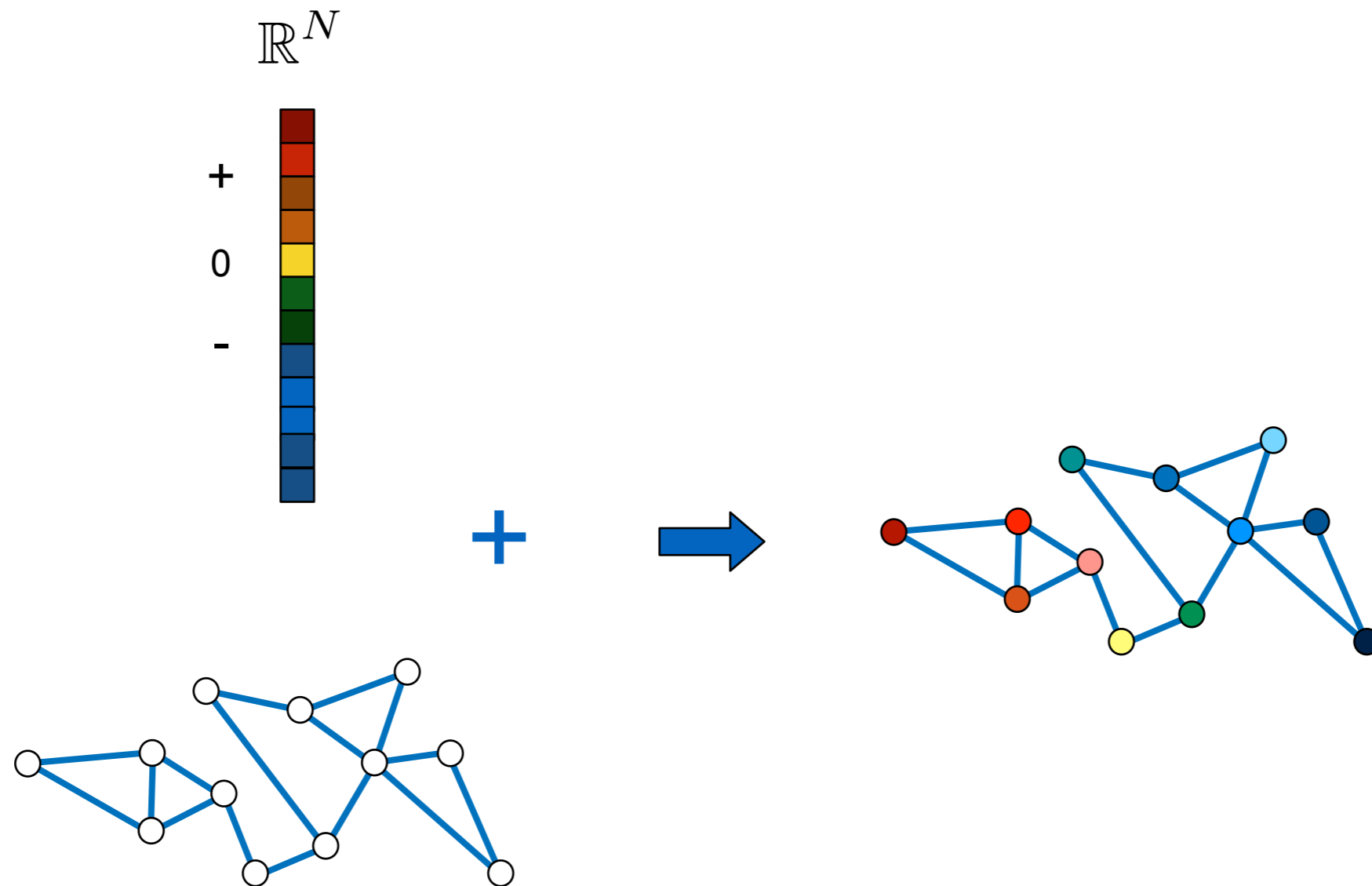


- Weighted graphs capture the geometric structure of complex, i.e., irregular, domains



# Processing graph structured data

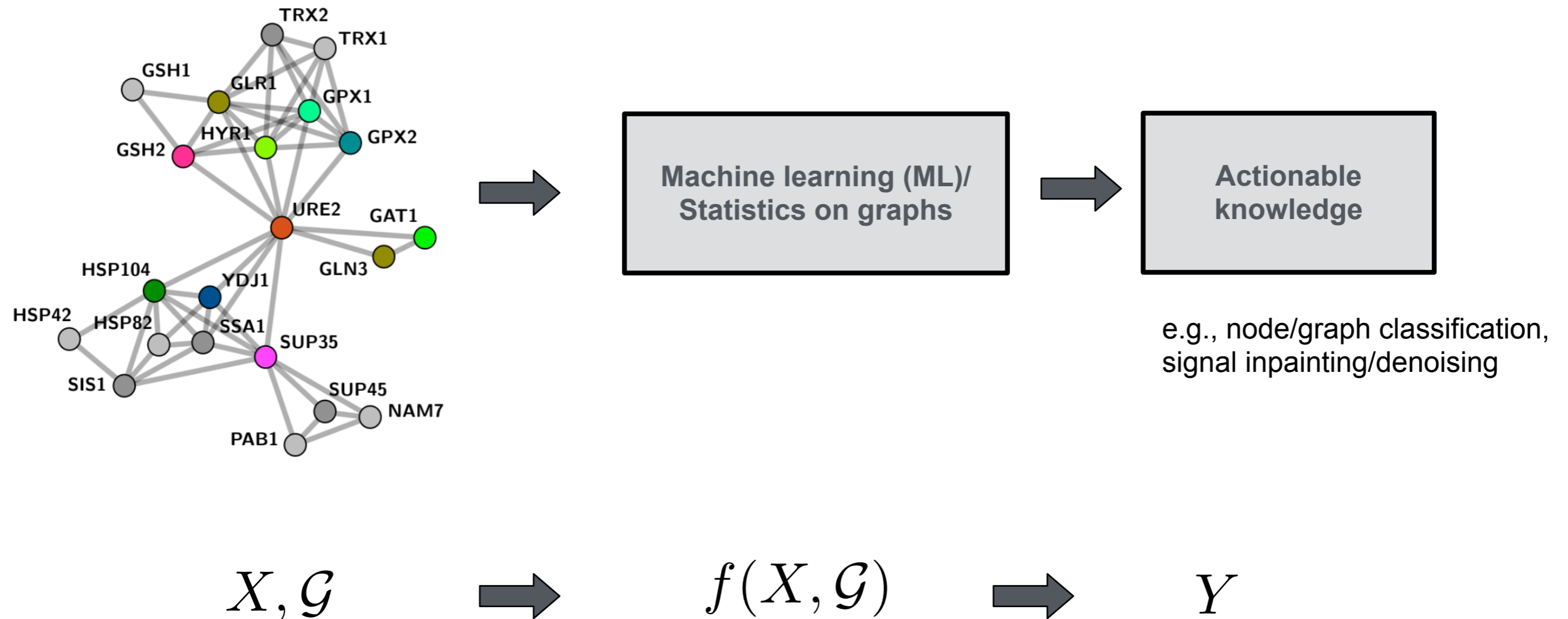
---



How can we extract useful information by taking into account both **structure (edges)** and **data (values/features on vertices)**?

# In this lecture...

- How can we infer useful information from graph structured data?



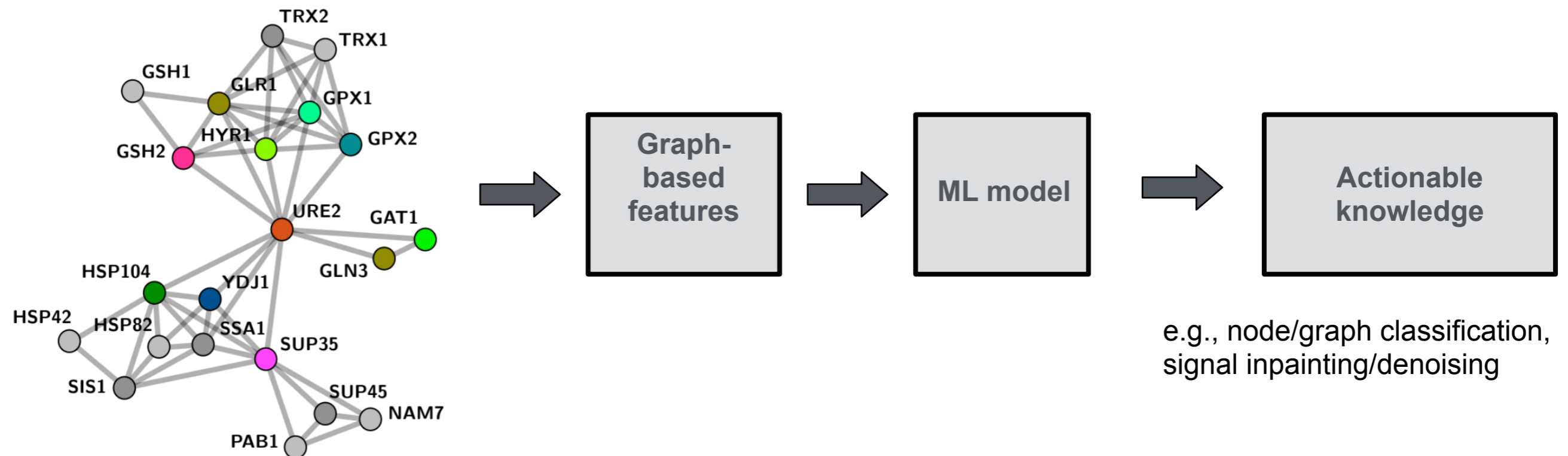
# Outline

---

- Traditional ML on graphs
  - Graph-based feature engineering
- Recent ML on graphs
  - Feature learning on graphs

# Traditional ML pipeline on graphs

- How can we learn useful information from graph structured data?



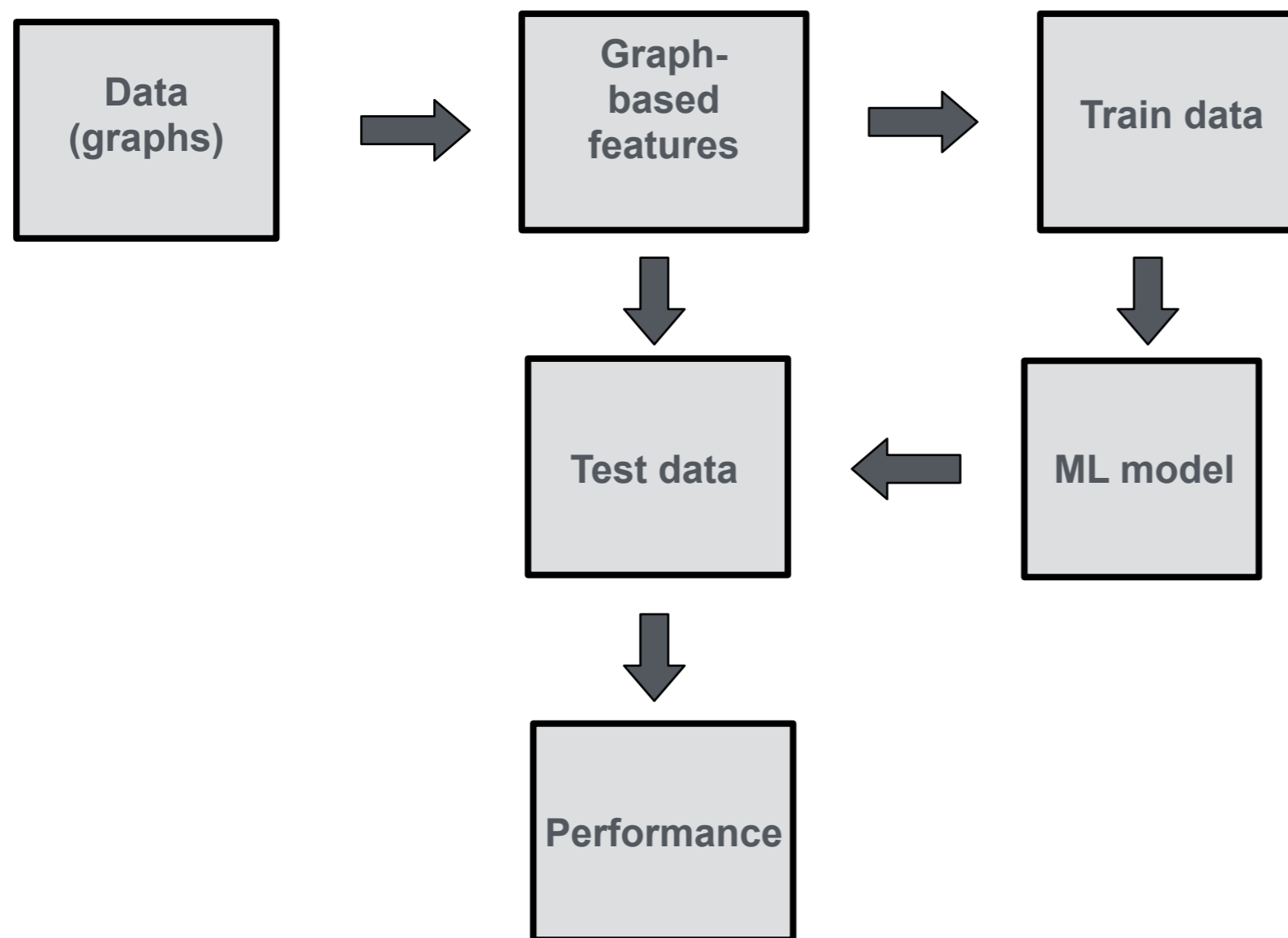
$$X, \mathcal{G} \quad \longrightarrow \quad \phi(X, \mathcal{G}) \quad \longrightarrow \quad f(\phi(X, \mathcal{G})) \quad \longrightarrow \quad Y$$

# Traditional ML pipeline on graphs

---

- Feature engineering is a way of extracting meaningful information from graphs

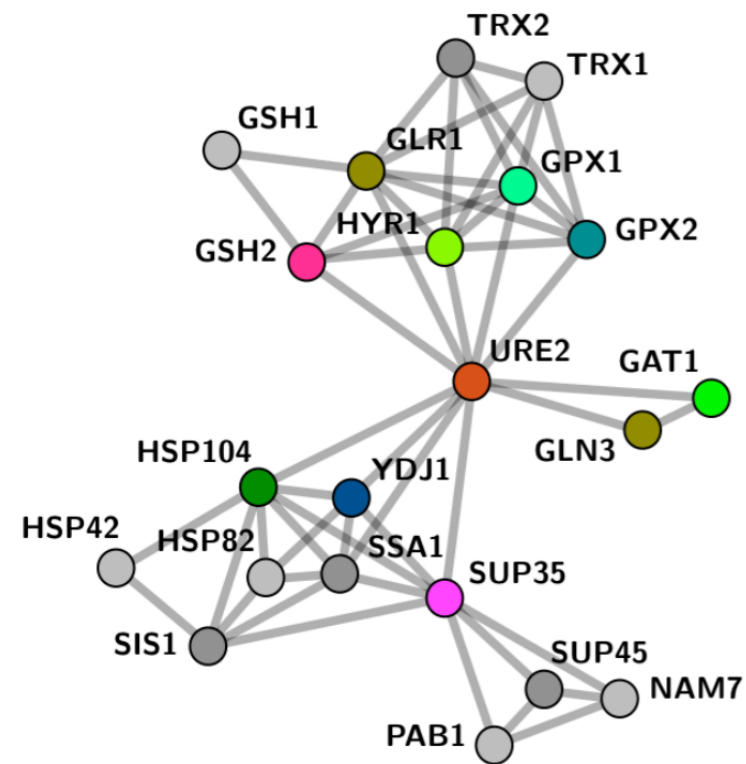
Feature engineering





# Traditional ML pipeline: Input

---



- Input:
  - Graph:  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
  - Graph with attributes:  $\mathcal{G}, X$

$X, \mathcal{G}$

# Traditional ML pipeline: Features

---

- Should reveal important information regarding the graph structure
- Key to achieving good model performance
- Features can be defined at different scales
  - At a node, edge, sets of nodes, entire graph level
- The choice of the features depends on
  - the end task
  - prior knowledge on the data

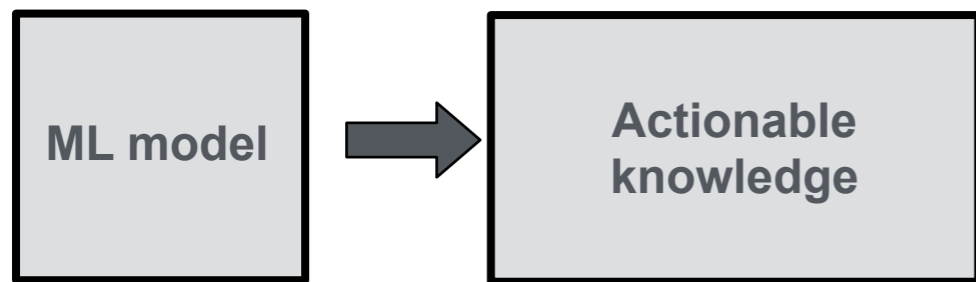
Graph-  
based  
features

$$\phi(X, \mathcal{G})$$

# Traditional ML pipeline: Learning tasks

---

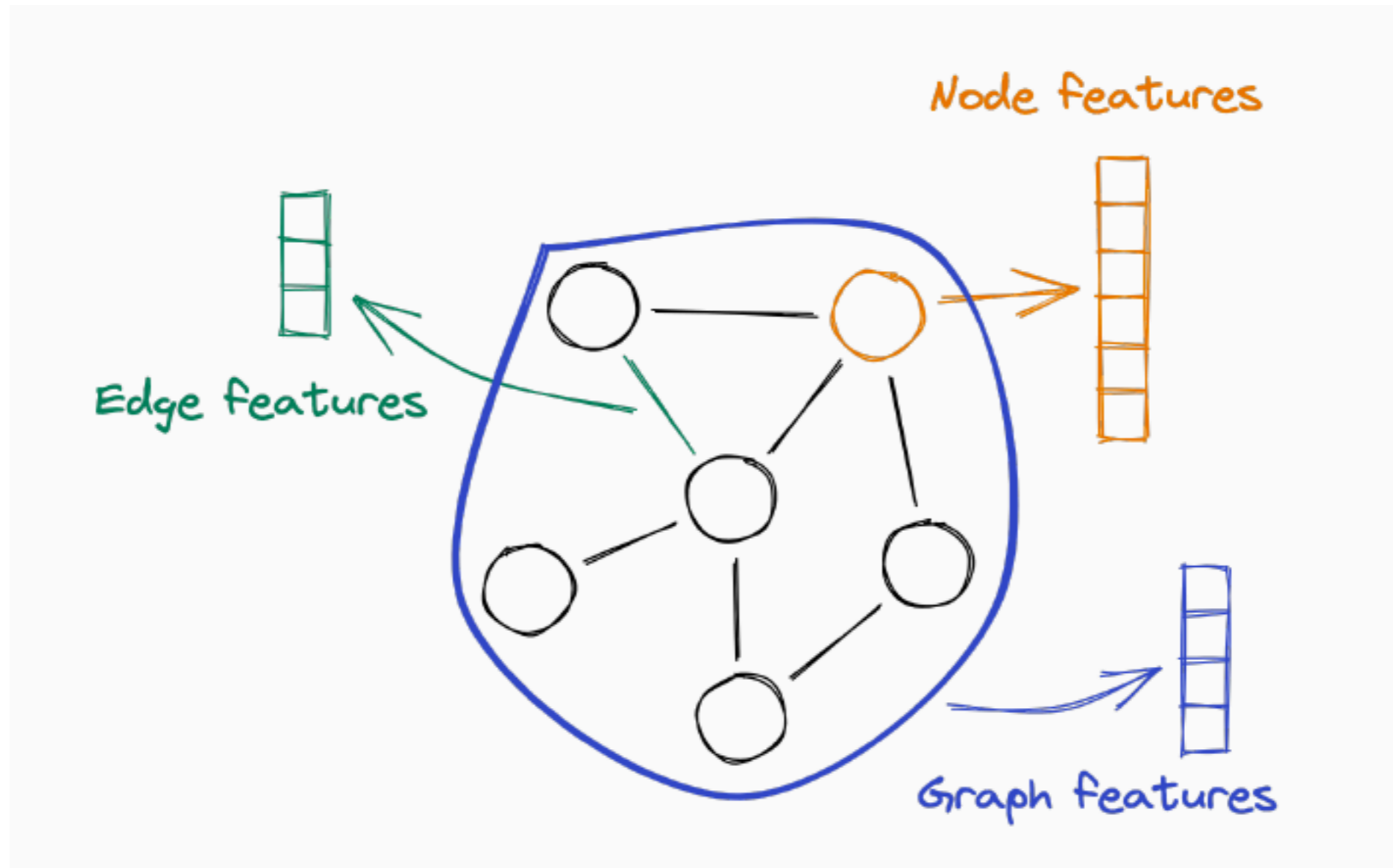
- The features are given as input to an ML model
  - Examples: logistic regression, SVM, neural networks, etc.
- Training phase:
  - Given a set of graph-based features, train a model  $f$  that predicts the correct  $Y$
- Testing phase:
  - Given a new node/link/graph, compute its features, and give them as an input to  $f$  to make a prediction



e.g., node/graph classification,  
signal inpainting/denoising

$$f(\phi(X, \mathcal{G})) \longrightarrow Y$$

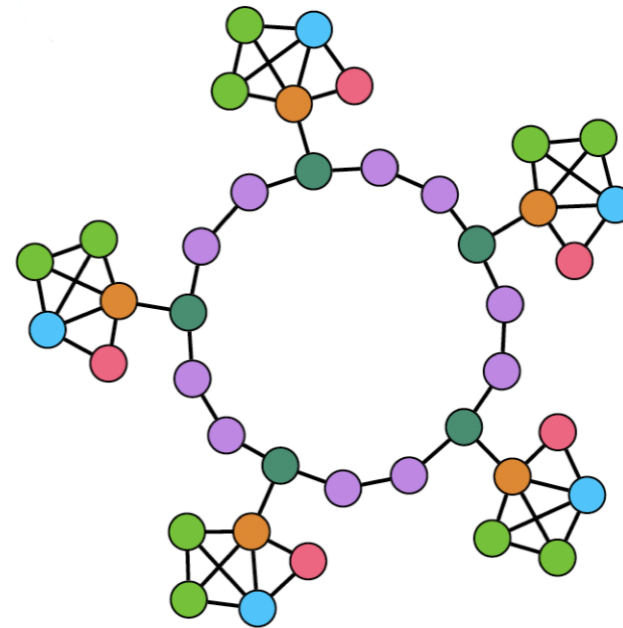
# Extracting information at different levels



# Node level features

---

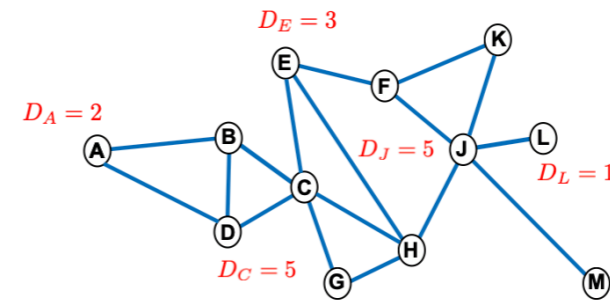
- Typically useful for node classification/clustering tasks



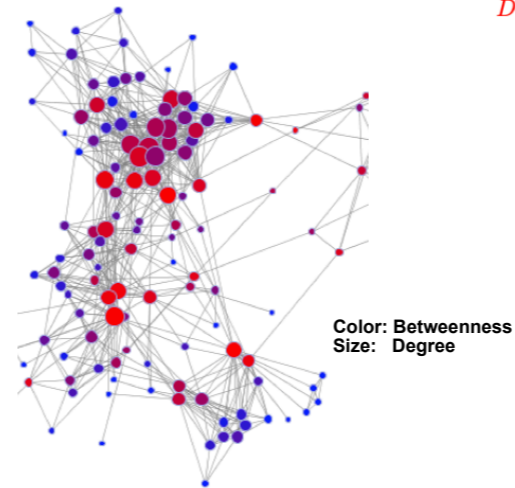
- Aim at characterizing the structure and position of a node in the network

# Common node level features

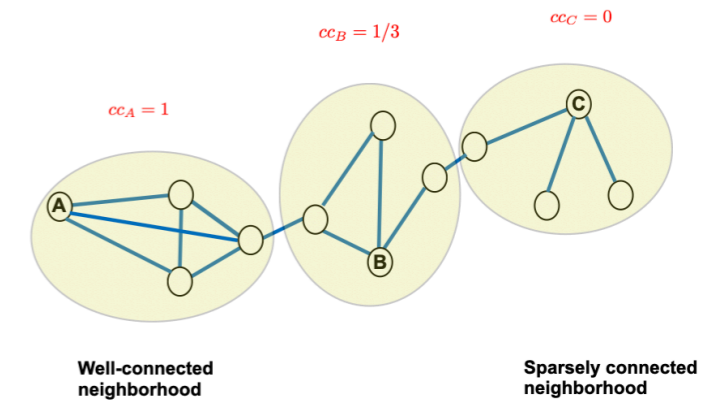
- Node degree



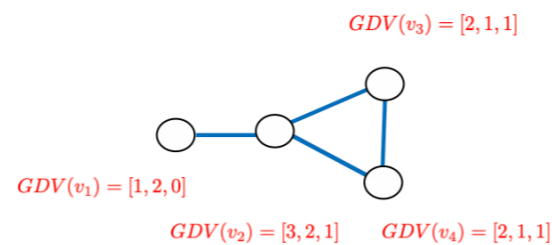
- Node centrality



- Clustering coefficient



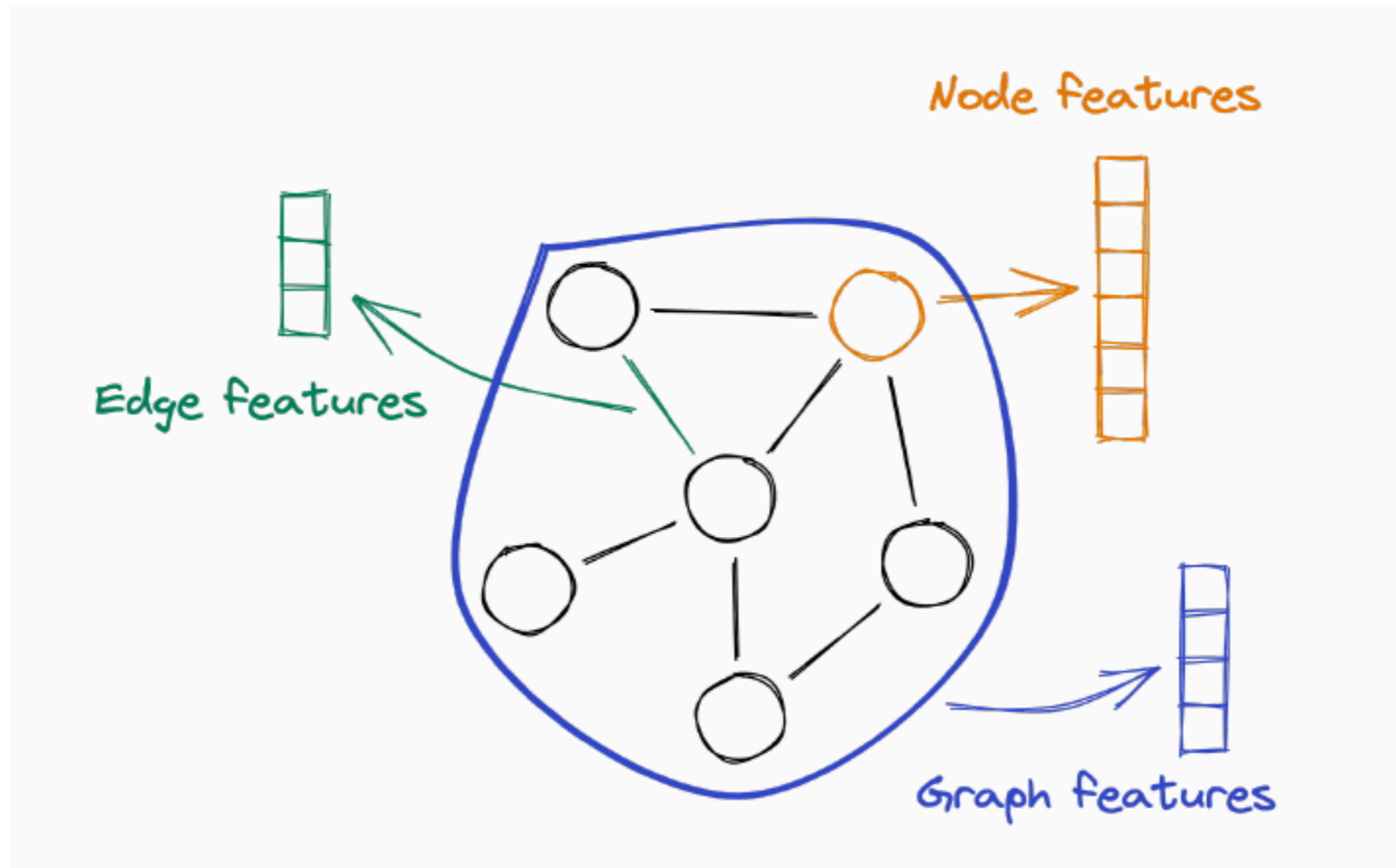
- Graphlets



Possible graphlets:



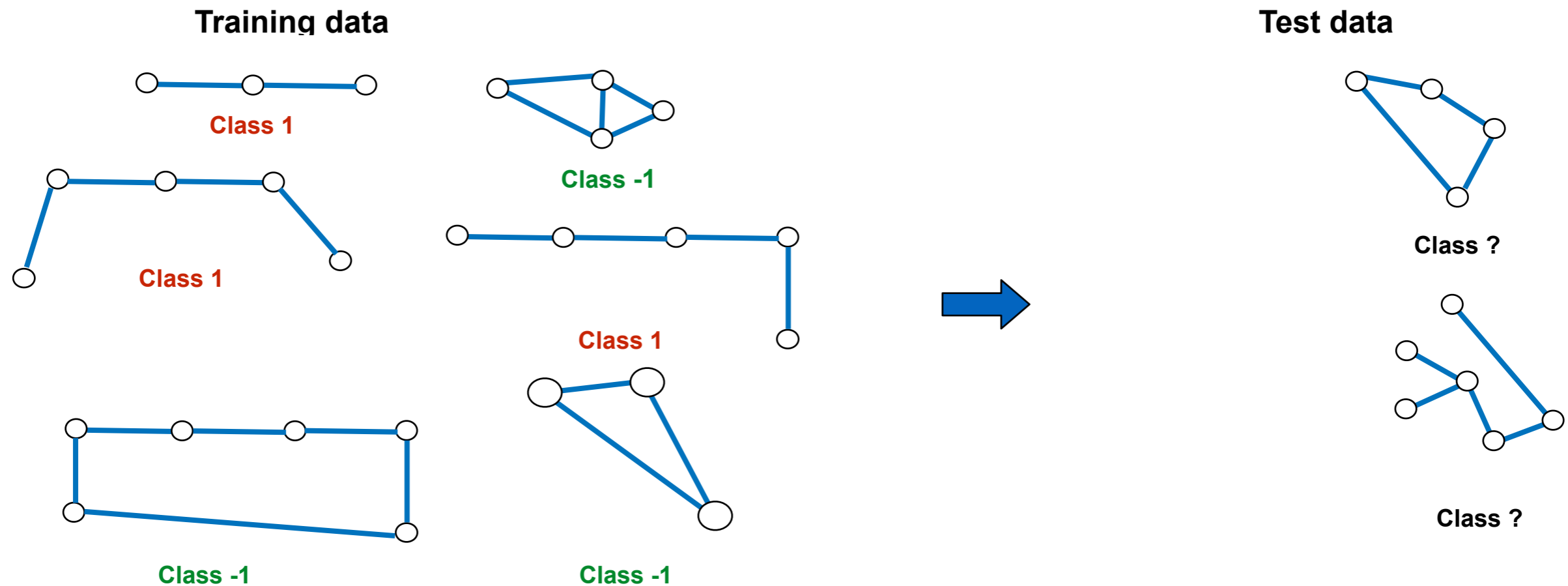
# From node level to graph level task



How can we design features that characterize the structure of the entire graph?

# Illustrative example: Graph classification

- Common assumption: Graphs with similar structure have similar label

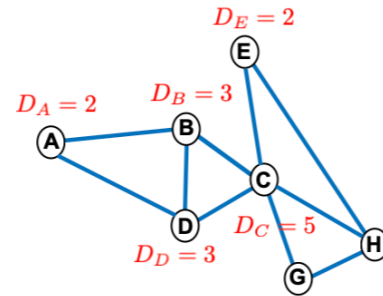


What is a good similarity metric between graphs?

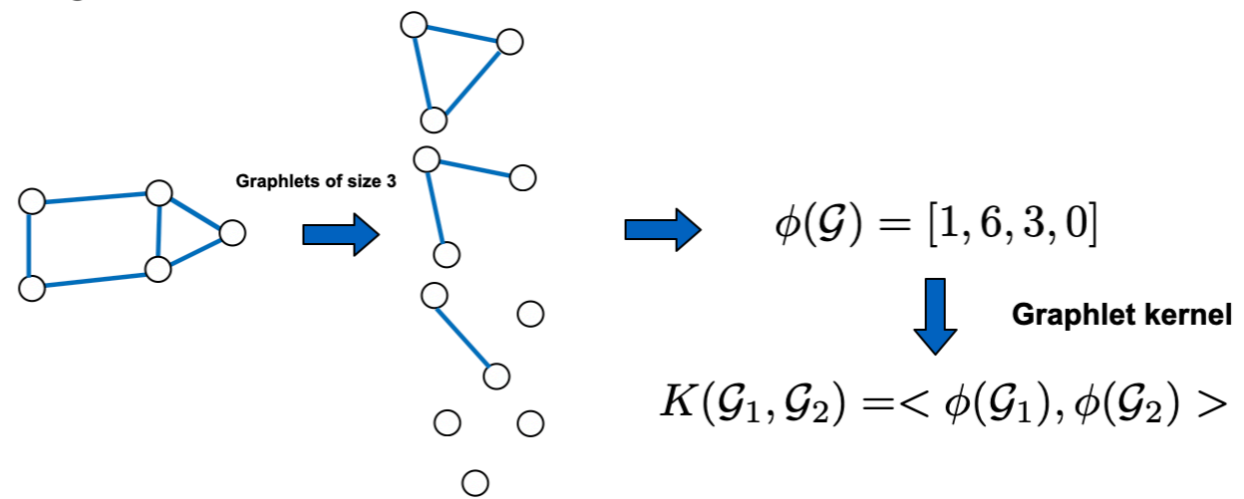


# Graph level features

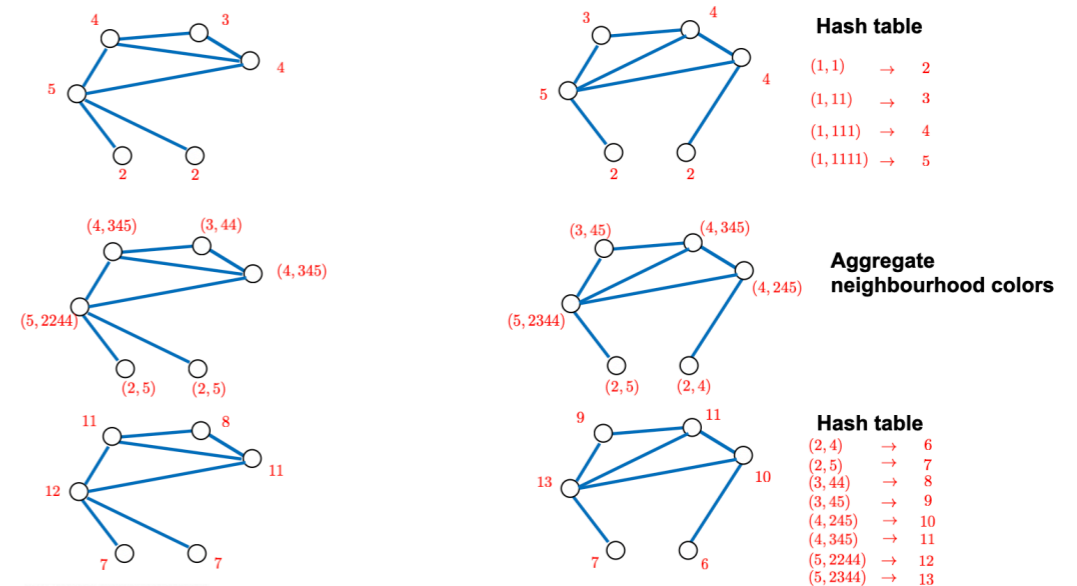
- Bag of nodes



- Graphlet kernel



- The Weisfeiler-Lehman kernel



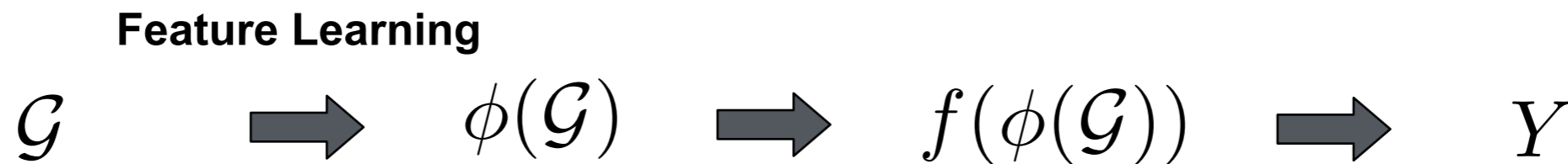
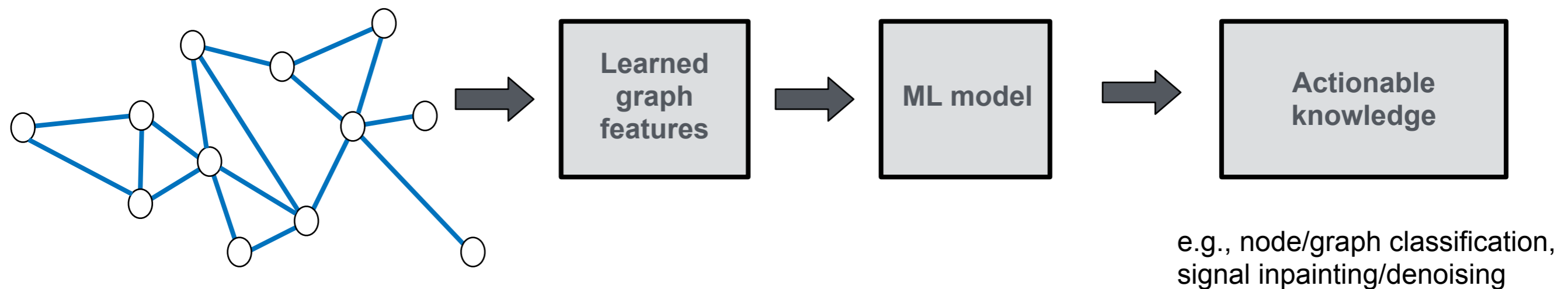
# Limitations of hand-crafted graph features

---

- Hand-engineered features are defined a priori: no adaptation to the data
- Designing graph features can very often can be a time consuming and expensive process
- Not easy to incorporate additional features on the nodes
- More flexibility can be achieved with an end-to-end learning pipeline

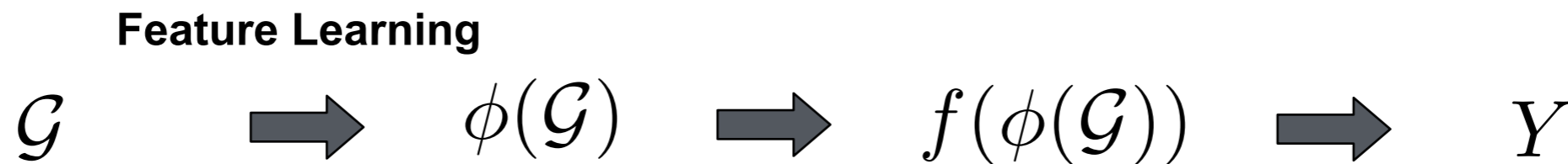
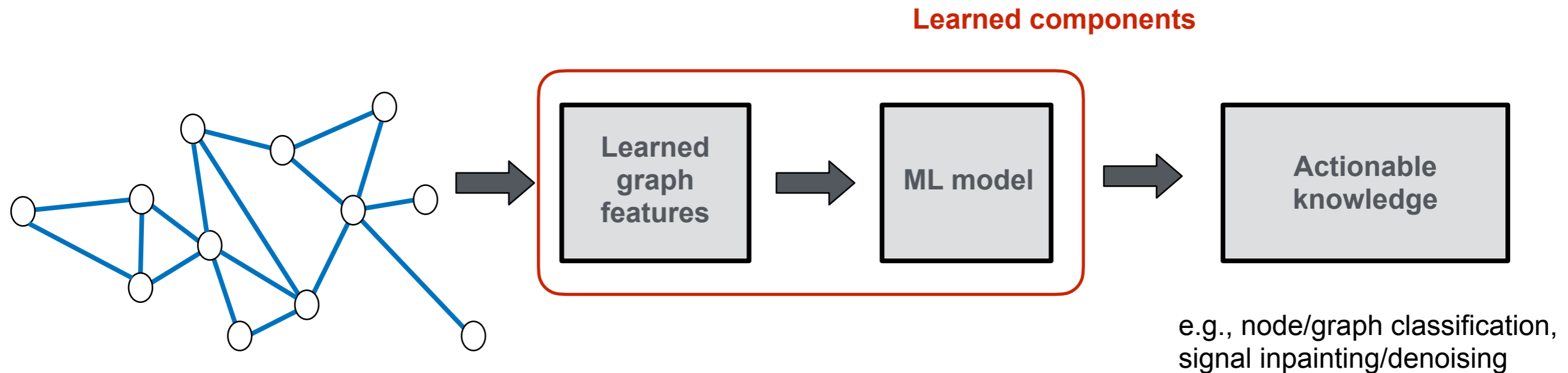
# Graph representation learning

- **Intuition:** Optimize the feature extraction part by adapting it to the specific instances of the graphs/data



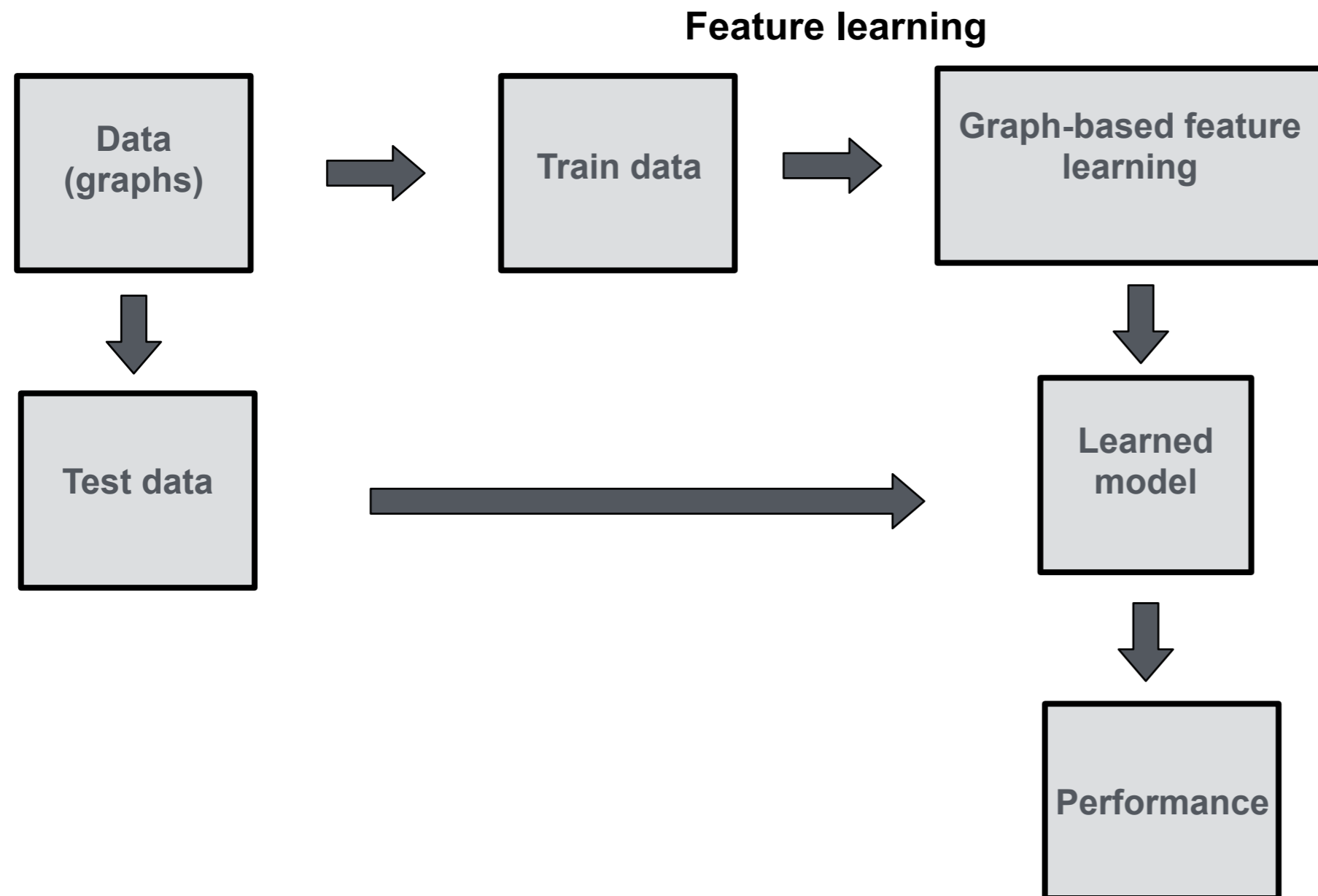
# Graph representation learning

- **Intuition:** Optimize the feature extraction part by adapting it to the specific instances of the graphs/data



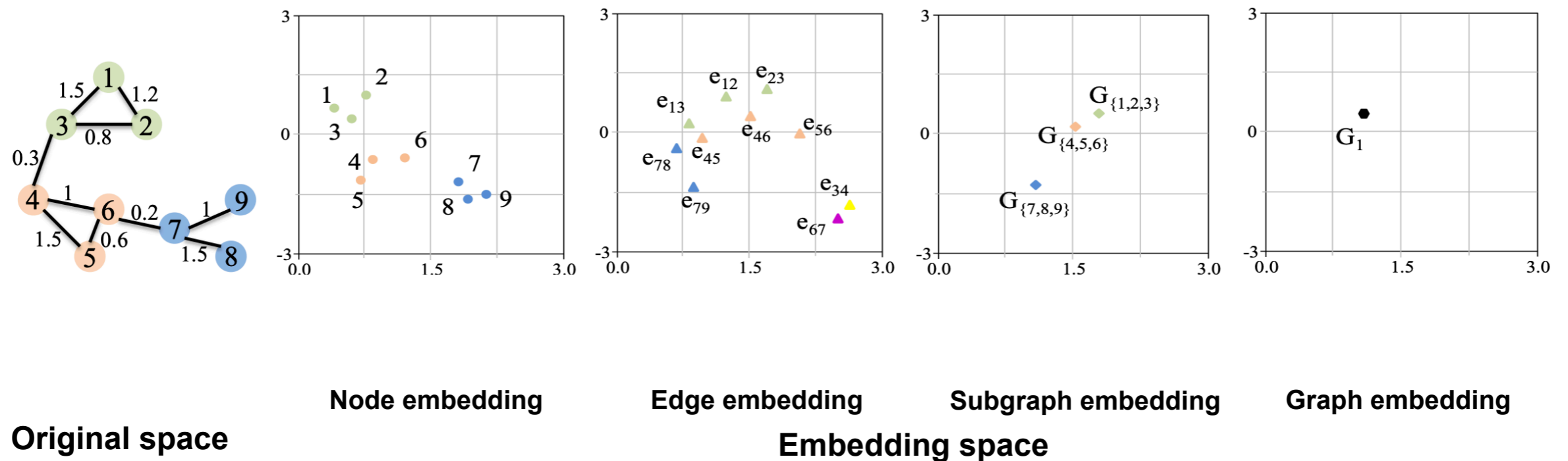
# Graph representation learning: basic pipeline

- Feature learning is a way of extracting data-adaptive graph representations



# Learning features on graphs

- Learned features convert the graph data in a (low dimensional) latent space (i.e., **embedding space**) where hidden/discriminative information about data is revealed

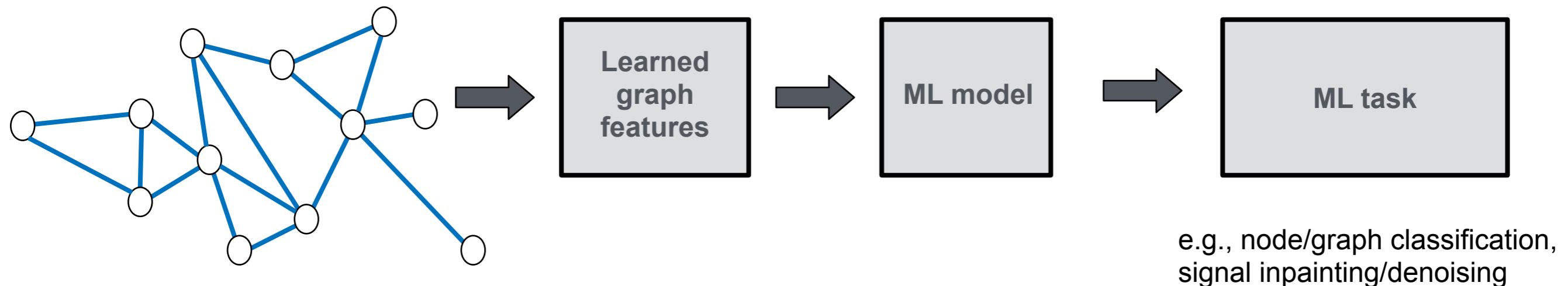


How can we learn the embedding space?

# Supervised graph representation learning

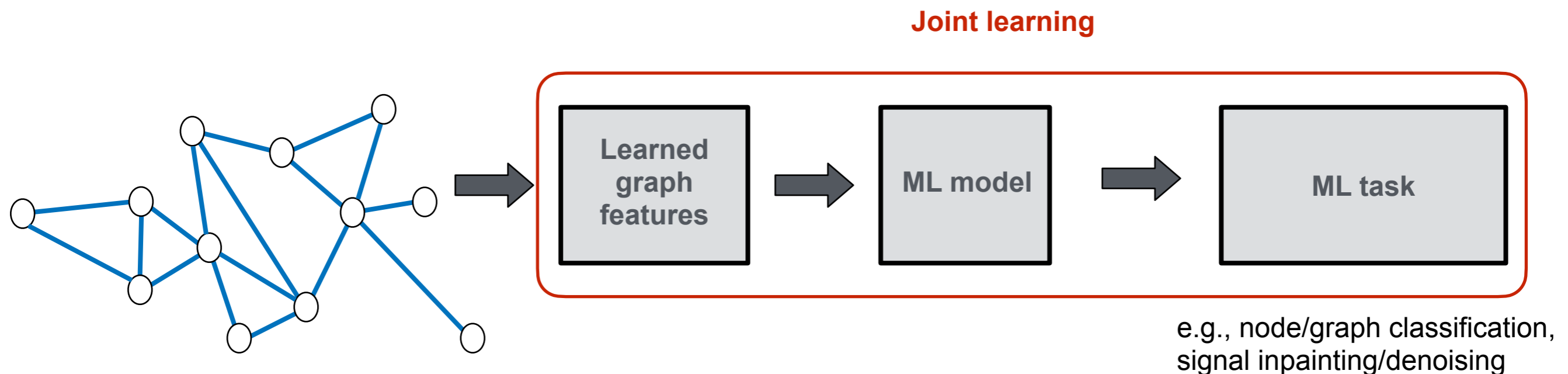
---

- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification



# Supervised graph representation learning

- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification





# Unsupervised graph representation learning

---

- Representations are not optimized for a specific downstream task
  - They are optimized with respect to some notion of “closeness” in the graph
  - The notion of “closeness” defines the design of the embedding algorithm
- Potentially used for many downstream inference tasks

Learned embedding vector

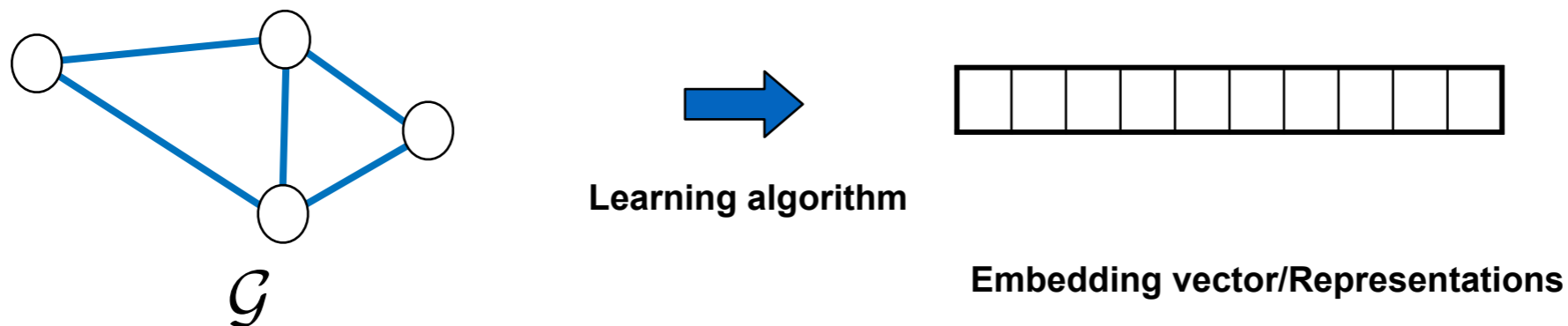


Example of tasks

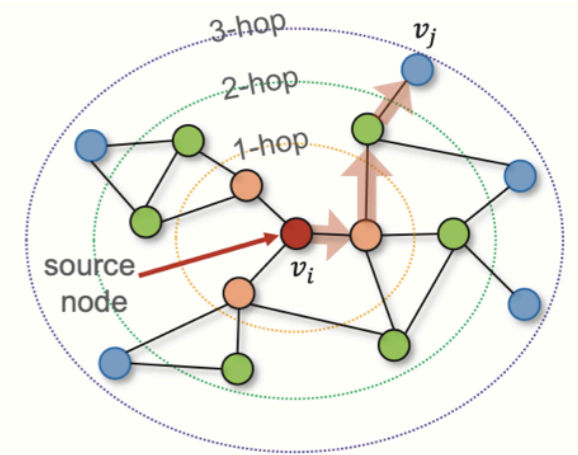
1. Node/graph classification
2. Node/graph clustering
3. Link prediction
4. Visualization
5. ...

# Embeddings on graphs: Definition

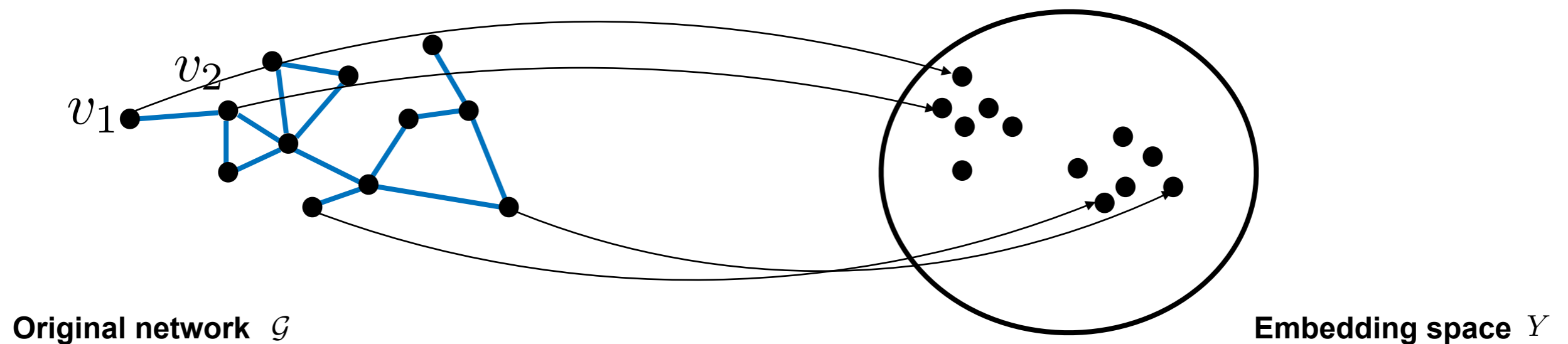
- Given an input graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ , and a predefined dimensionality of the embedding  $d \ll |\mathcal{V}|$ , the goal is to convert  $\mathcal{G}$  (or a subgraph, or a node) into a  $d$ -dimensional space in which graph properties are preserved



- Graph properties can be quantified using proximity measures on the graph (e.g.,  $K$ -hop neighborhood)



# Illustrative example: Node embeddings



What is the similarity in the graph that should be preserved in the embedding space?

$$\text{sim}_{\mathcal{G}}(v_1, v_2) \approx \text{sim}_Y(Y_1, Y_2)$$

# Example 1: Laplacian Eigenmaps

---

- **Intuition:** Preserve pairwise node similarities derived from the adjacency/weight matrix

$$sim_{\mathcal{G}}(v_i, v_j) = W_{ij}$$

- Measure similarity in the embedding space using the mean square error

$$sim_Y(Y_i, Y_j) = \|Y_i - Y_j\|_2^2$$

- Impose larger penalty if two nodes with larger pairwise similarity are embedded far apart

$$\begin{aligned} l(sim_{\mathcal{G}}(v_i, v_j), sim_Y(Y_i, Y_j)) &= sim_{\mathcal{G}}(v_i, v_j) \cdot sim_Y(Y_i, Y_j) \\ &= W_{ij} \|Y_i - Y_j\|_2^2 \end{aligned}$$

# Laplacian Eigenmaps - reminder

- Compute embeddings that minimize the expected square distance between connected nodes

Centered embeddings

Uncorrelated

embedding coordinates

$$\min_{Y \in \mathbb{R}^{N \times K} : Y^T \mathbf{1} = 0, Y^T Y = I_K} \sum_{(i,j) \in \mathcal{E}} W_{ij} \|Y_i - Y_j\|^2$$

⇓ Graph smoothness

$$\min_{Y \in \mathbb{R}^{N \times K} : Y^T \mathbf{1} = 0, Y^T Y = I_K} \text{tr}(Y^T L Y)$$

⇓ Lagrangian

$$\min_{Y \in \mathbb{R}^{N \times K} ; Y^T \mathbf{1} = 0} \text{tr}(Y^T L Y - (Y^T Y - I_K) \Gamma)$$

⇓ Gradient

$$L Y = Y \Gamma \Rightarrow u_i \rightarrow (\chi_2(i), \dots, \chi_{K+1}(i))$$

**Laplacian Eigenmaps:**  $K$  first non-trivial eigenvectors of the Laplacian!

[Belkin et al, 2003, Laplacian Eigenmaps for Dimensionality Reduction and Data Representation, Neural Comp.]

# Example 2: DeepWalk

---

- **Intuition:** Nodes have similar embeddings if they tend to cooccur on short random walks over the graph

$$sim_{\mathcal{G}}(v_i, v_j) = p(v_j|v_i)$$

- **Objective:** Given node  $v_i$  learn a mapping  $\phi : v_i \rightarrow \mathbb{R}^d$ ;  $\phi(v_i) = Y_i$  such that the feature representation  $Y_i$  are predictive of the nodes in its random walk neighborhood  $\mathcal{N}_{v_i|RW}$

$$\max_{\phi} \sum_{v_i \in \mathcal{V}} \log sim_Y(Y_i, Y_j) = \max_{\phi} \sum_{v_i \in \mathcal{V}} \log P(Y_j \text{ for } v_j \in R_{i|RW} | Y_i)$$

**Maximum likelihood**

- Measure the similarity in the embedding space in a probabilistic manner

$$sim_Y(Y_i, Y_j) = \frac{e^{Y_i^T Y_j}}{\sum_{k \in \mathcal{V}} e^{Y_i^T Y_k}}$$

**Softmax**

# DeepWalk - Algorithm

---

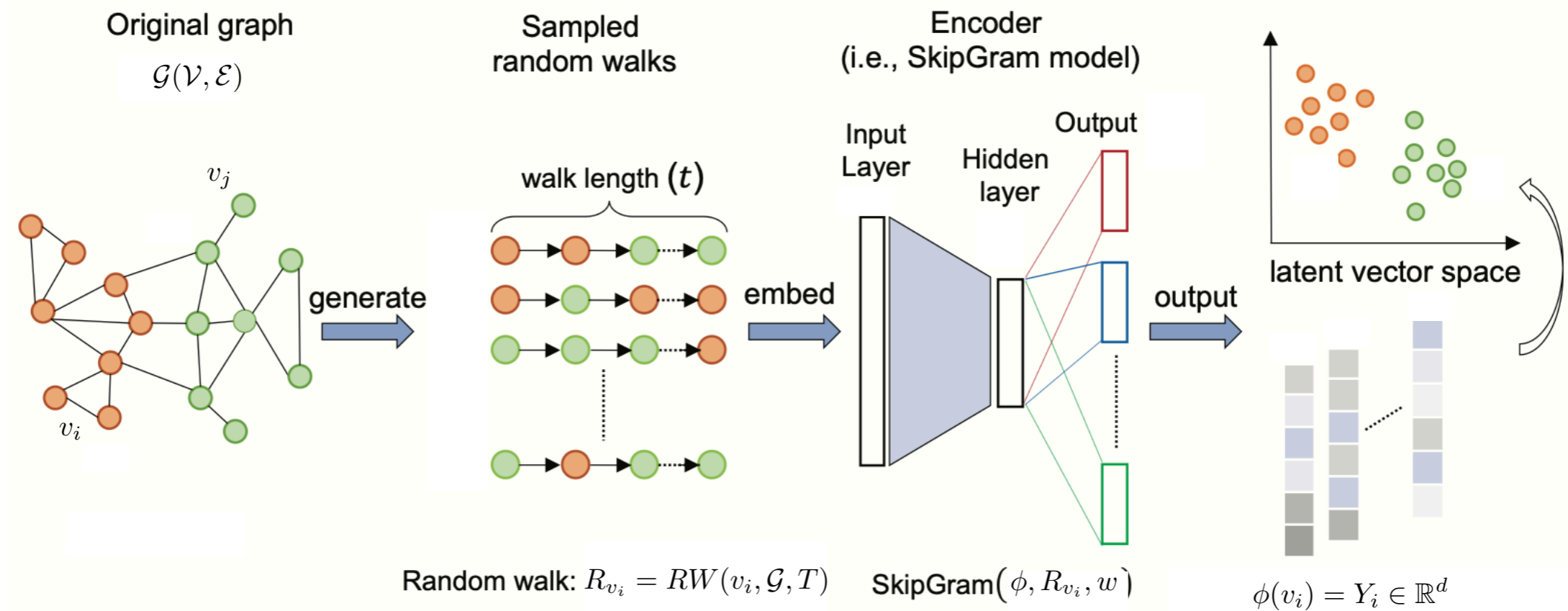
- Run fixed length random walks starting from each node of the graph
- For each node  $v_i$  define its random walk neighborhood  $\mathcal{N}_{v_i|RW}$
- Find embeddings to maximize the likelihood of random walk co-occurrences

$$loss_{(v_i, v_j) \in N_{Train}} = \sum_{v_i \in N_{Train}} \sum_{v_j \in R_{v_i|RW}} -\log \left( \frac{e^{Y_i^T Y_j}}{\sum_{v_k \in N_{Train}} e^{Y_i^T Y_k}} \right)$$

Predicted probability of two nodes co-occurring in a random walk

- Embeddings are optimized using stochastic gradient descent

# DeepWalk - Schematic overview

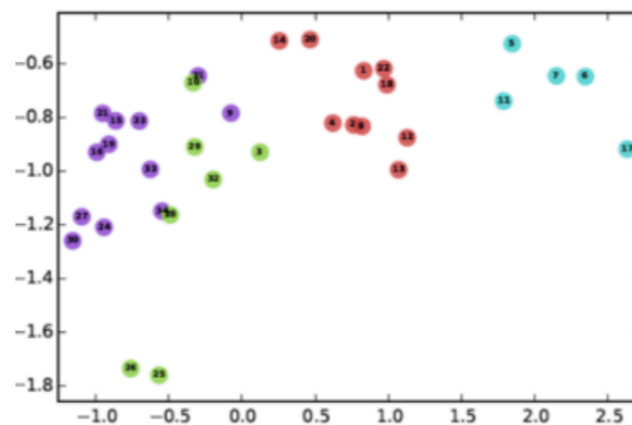
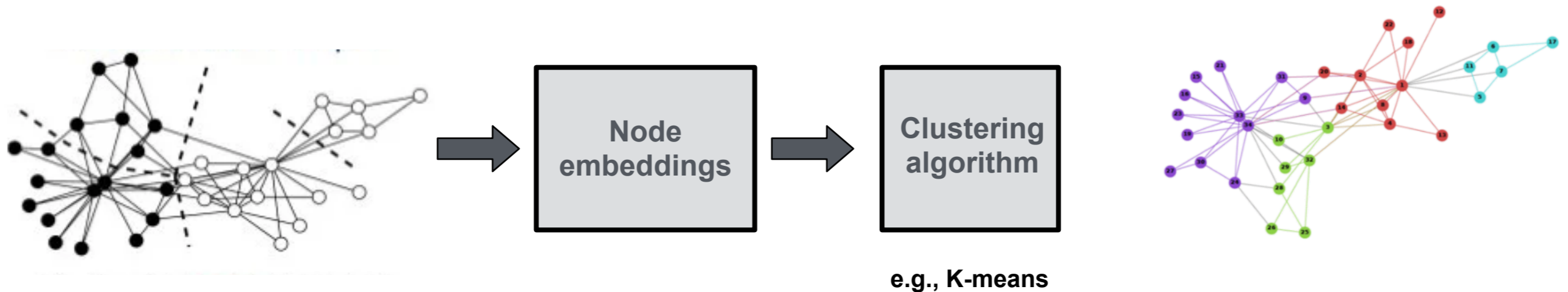


[Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD*]



# Application of node embeddings: Node clustering/Community detection

- The karate-club example
  - Compute node embeddings
  - Apply any clustering algorithm (e.g., K-means) on the learned embeddings



# Summary so far

---

- Feature learning on graphs is a data-driven (and often more flexible) alternative to designing hand-crafted features
- Unsupervised learning on graphs provides representations i.e., embeddings, that are not adapted to specific tasks
- Different assumptions lead to different ways of preserving information from the original graph in the embedding space (e.g., weight matrix, random walks...)
- The choice of what structure information to preserve depends on the application

# Limitations of the (discussed) node embedding algorithms

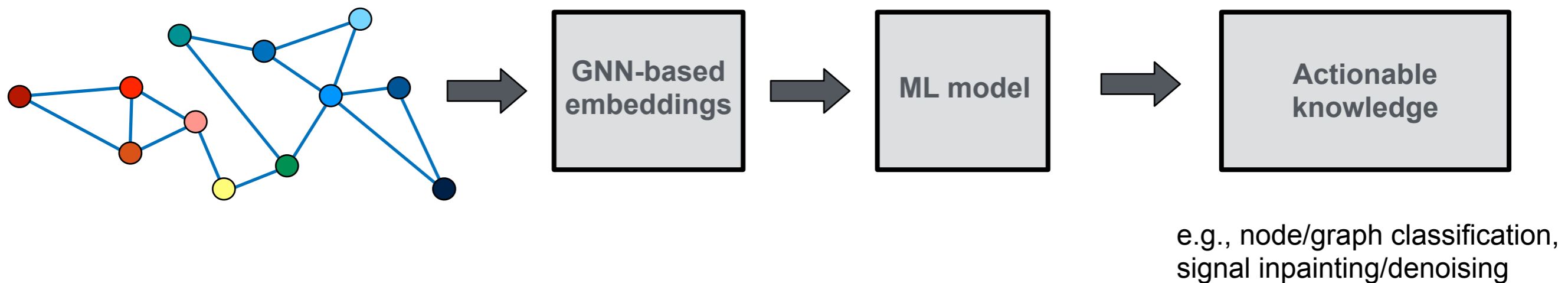
---

- Usually transductive not inductive
  - Learned embedding models often do not generalize to new nodes
- Do not incorporate node attributes
- Independent of downstream tasks
- No parameter sharing:
  - Every node has its own unique embedding

# Graph neural networks (GNNs)

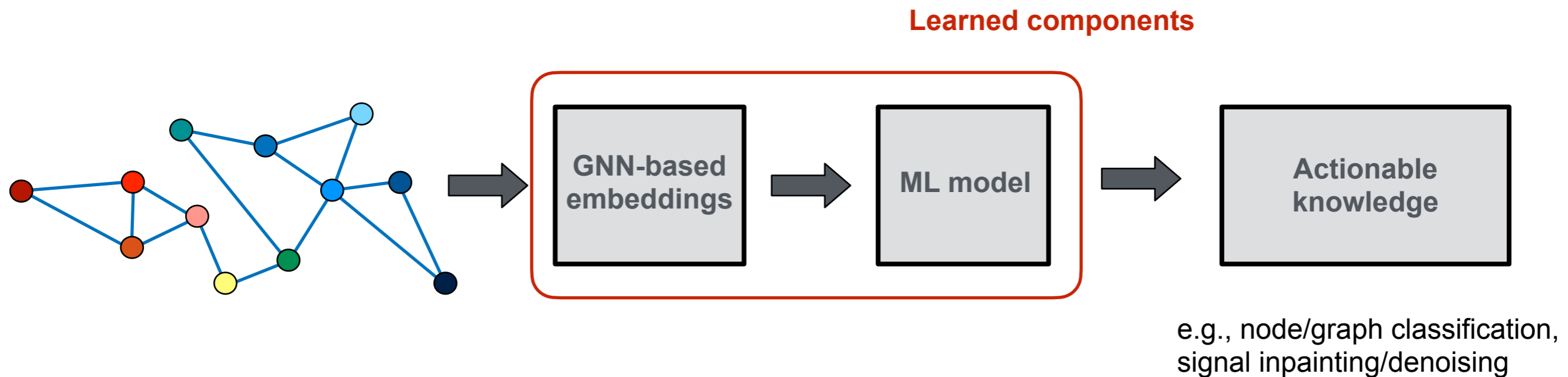
---

- A different way of obtaining ‘deeper’ embeddings inspired by deep learning
- They generalize to graphs with node attributes



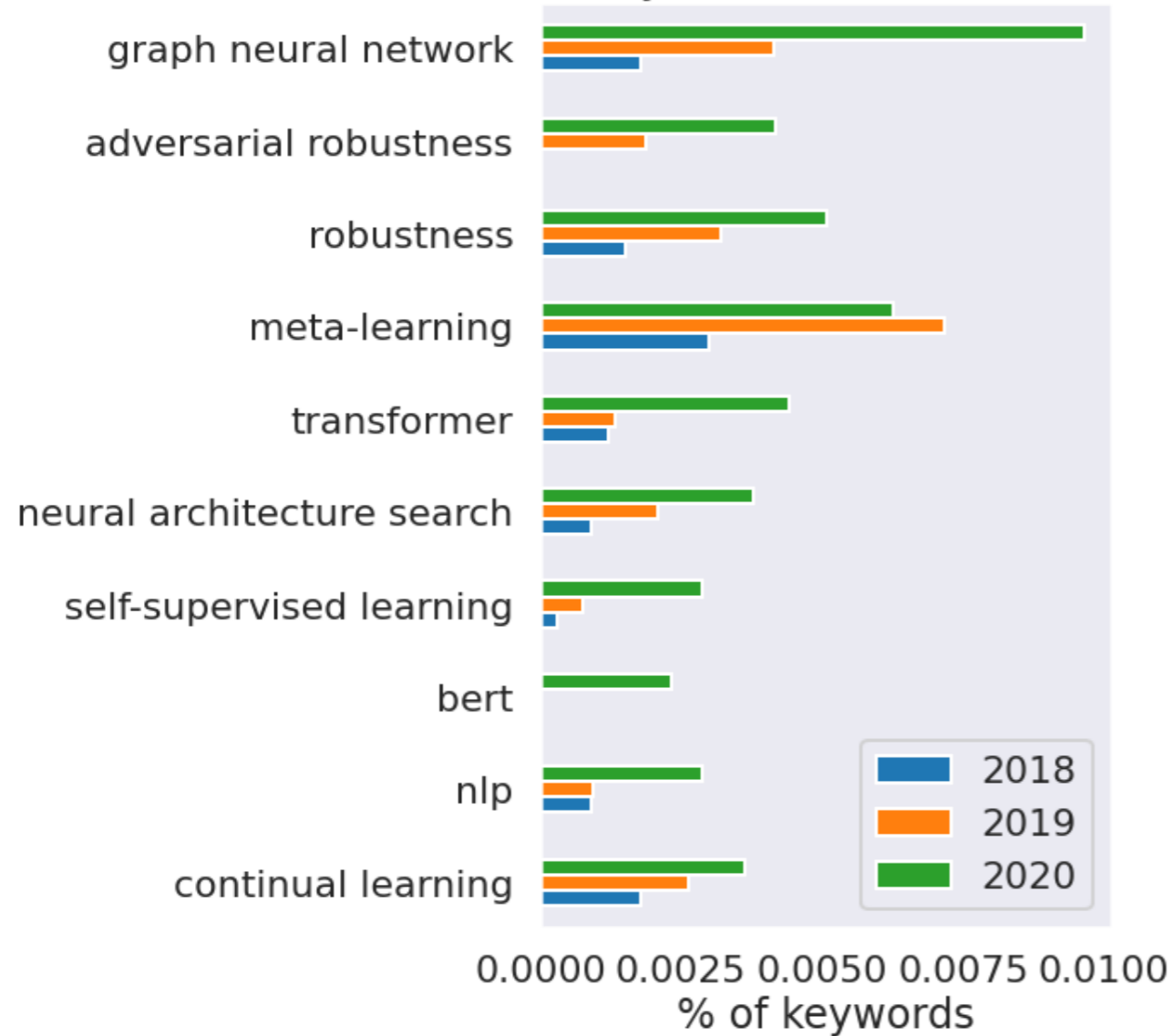
# Graph neural networks (GNNs)

- A different way of obtaining ‘deeper’ embeddings inspired by deep learning
- They generalize to graphs with node attributes

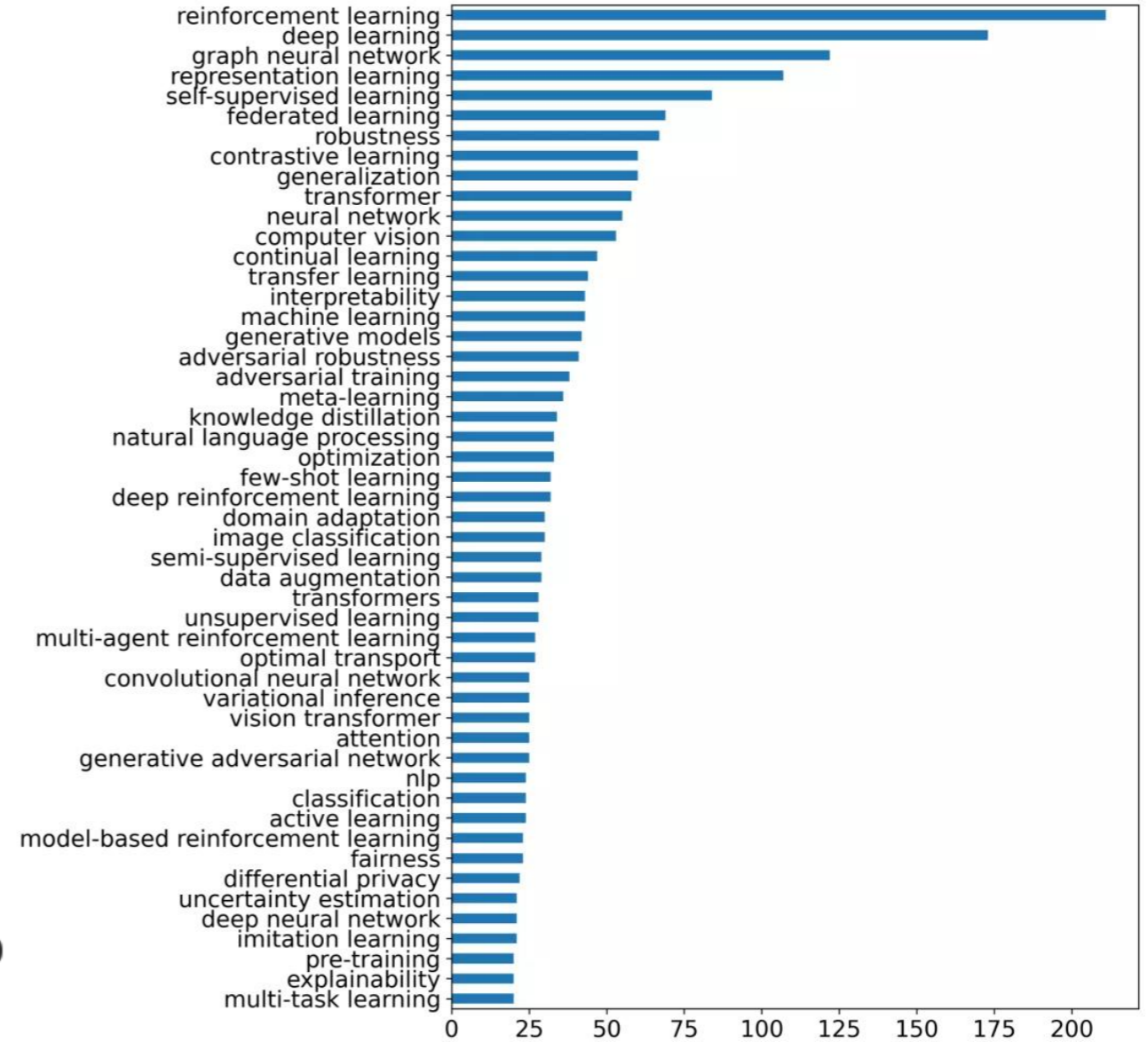


# GNNs: A growing trend

ICLR Keyword Growth 2018-2020



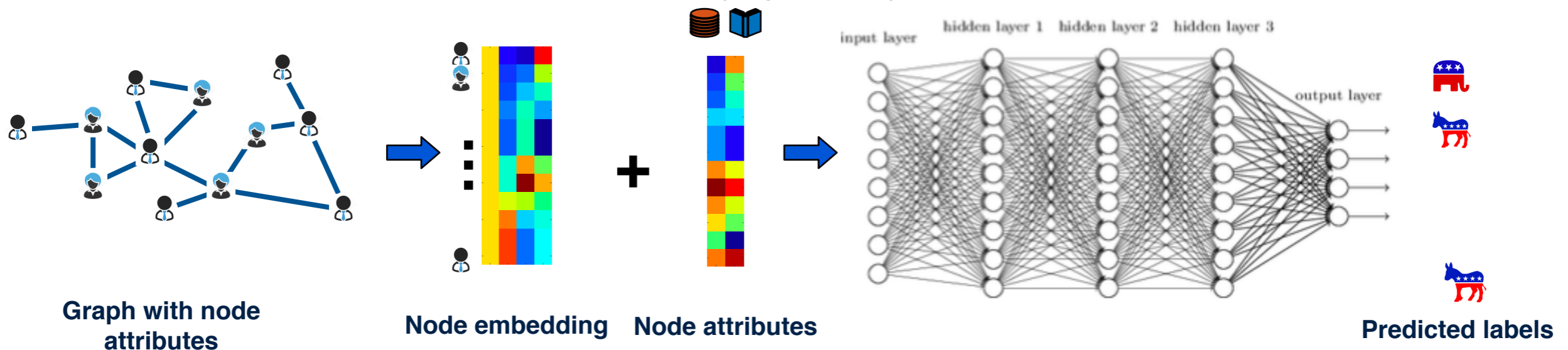
50 MOST APPEARED KEYWORDS



ICLR 2022

# Computing embeddings from graphs with node attributes

- A naive approach:
  - Embed graph and node attributes into a Euclidean space
  - Feed them into a deep neural net (e.g., MLP)



- Issues with that:
  - Computationally expensive
  - Not applicable to graphs of different sizes
  - Not invariant to node ordering: if we reorder nodes the representations will be different

**Can we do better? Yes!**

# Good priors are key to learning

---

- We build intuition from classical deep learning algorithms
- CNNs exploit structure in the images

Translation invariance



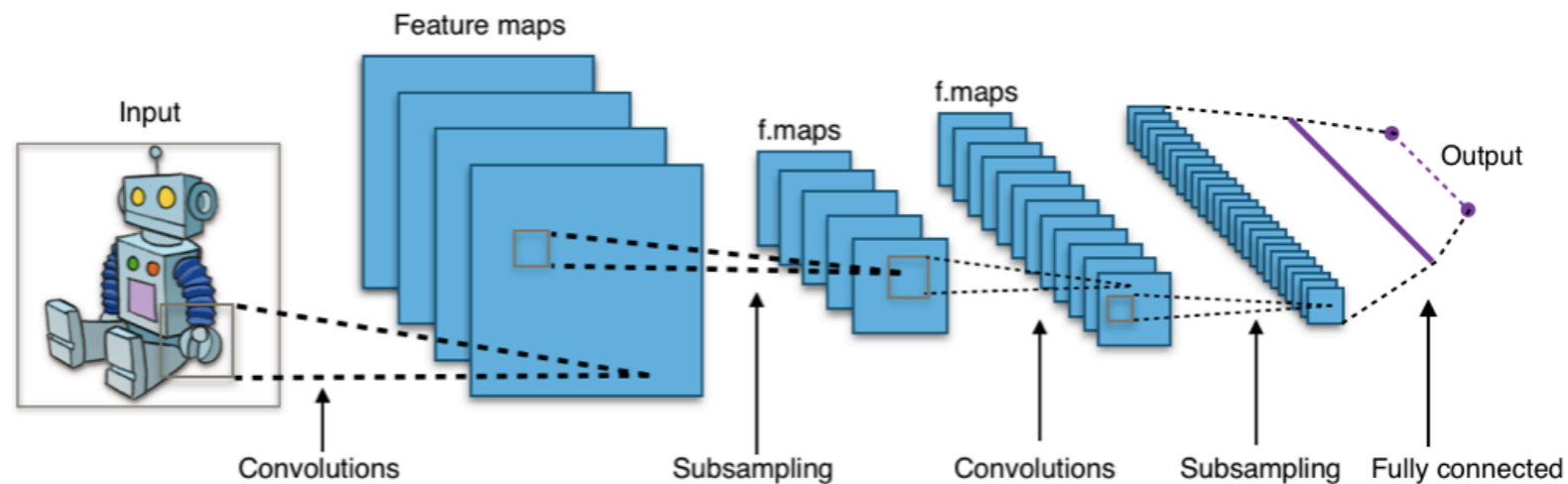
Composability





# CNN architecture: Illustrative example

- CNNs hierarchically aggregate (through convolution) and pool (i.e., subsample) images along pixel-grid



[https://en.wikipedia.org/wiki/File:Typical\\_cnn.png](https://en.wikipedia.org/wiki/File:Typical_cnn.png)

# How can we extend CNNs on graphs?

---

- Desirable properties
  - **Convolution:** how to achieve translation invariance
  - **Localization:** what is the notion of locality
  - **Graph pooling:** how to downsample on graphs
  - **Efficiency:** how to keep the computational complexity low
  - **Generalization:** how to build models that generalize to unseen graphs

# Towards a convolution on graphs: A spectral viewpoint

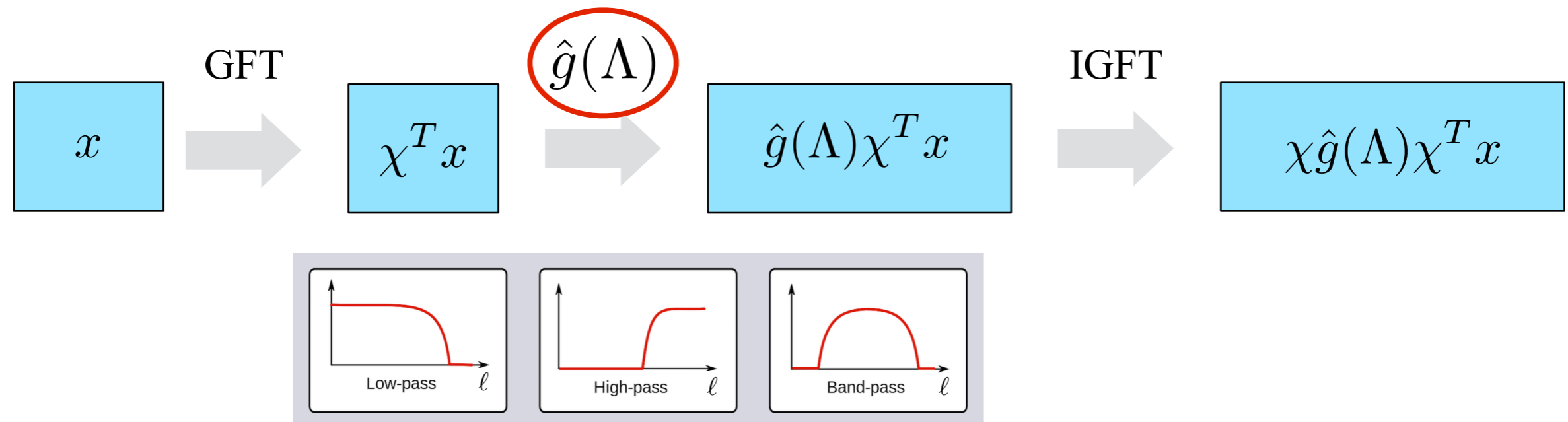
- **Key intuition:** Convolution in the vertex domain is equivalent to multiplication in the spectral domain
- Recall that: The graph Fourier transform of a graph signal  $x$  is defined using the eigenvectors and the eigenvalues of the Laplacian matrix ( $L = \chi\Lambda\chi^T$ )
- We define convolution on graphs starting from the multiplication in the GFT domain

<p style="color: red; font-weight: bold;">Classical convolution</p> $(x * g)(t) = \int_{-\infty}^{\infty} x(t - \tau)g(\tau)d\tau$ <p style="text-align: center; color: red; font-weight: bold;">FT</p> <p style="text-align: center; color: gray; font-size: 2em;">↓</p> $\widehat{(x * g)}(\omega) = \hat{x}(\omega) \cdot \hat{g}(\omega)$	<p style="color: red; font-weight: bold;">Convolution on graphs</p> $x * g = \chi\hat{g}(\Lambda)\chi^T x = \hat{g}(L)x$ <p style="text-align: center; color: gray; font-size: 2em;">↑</p> <p style="text-align: center; color: red; font-weight: bold;">IGFT</p>	$\widehat{(x * g)}(\lambda) = ((\chi^T x) \circ \hat{g})(\lambda)$ <p style="text-align: right; color: red; font-weight: bold;">GFT</p>
---	---	---

How can we interpret graph convolution?

# Reminder: Graph spectral filtering

- It is defined in direct analogy with classical filtering in the frequency domain
  - Filtering a graph signal  $x$  with a spectral filter  $\hat{g}(\cdot)$  is performed in the graph Fourier domain



**Convolution on graphs is equivalent to filtering!**

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L) x$$

Shuman et al., "The emerging field of signal processing on graphs", IEEE Signal Process. Mag., 2013

# Is the graph convolution localized?

---

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L)x$$

# Is the graph convolution localized?

---

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L)x$$

**Example:**  $\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \theta \in \mathbb{R}^{K+1}$   $\longrightarrow$   $\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$

# Is the graph convolution localized?

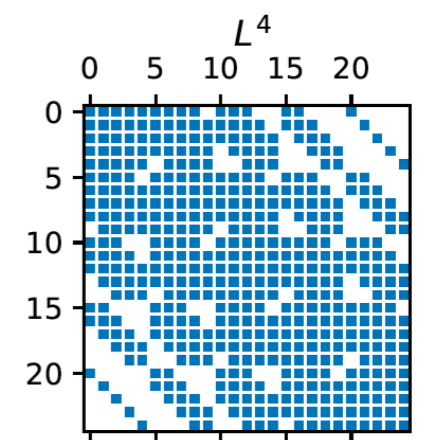
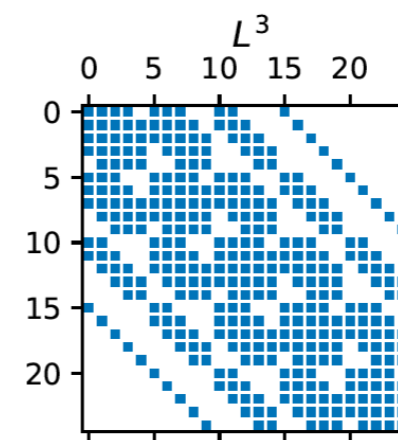
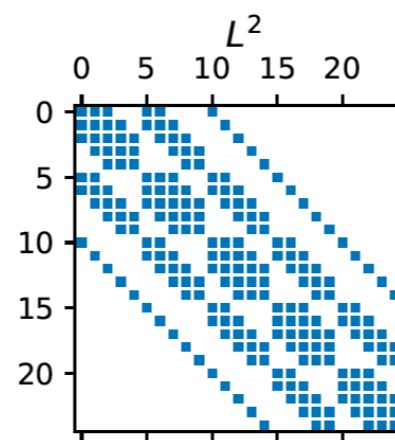
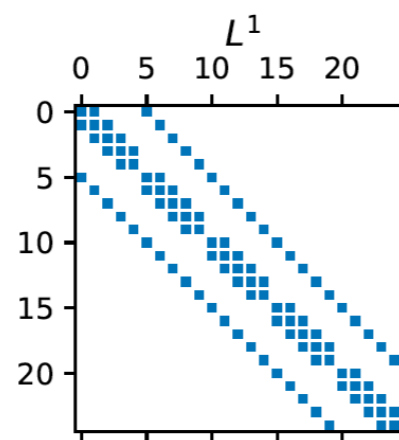
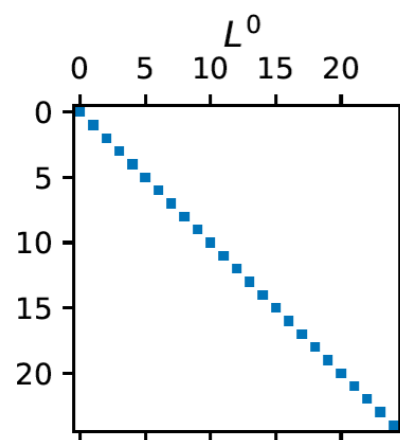
- In general the answer is no!

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L)x$$

- However, if we consider polynomial filters, the answer is yes

**Example:**  $\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \theta \in \mathbb{R}^{K+1} \quad \longrightarrow \quad \hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$

- $L^K$  defines the  $K$ -hop neighborhood:  $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



# Is the graph convolution localized?

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L)x$$

**Example:**  $\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \theta \in \mathbb{R}^{K+1}$   $\longrightarrow$   $\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$

- $L^K$  defines the  $K$ -hop neighborhood:  $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



# Is the graph convolution localized?

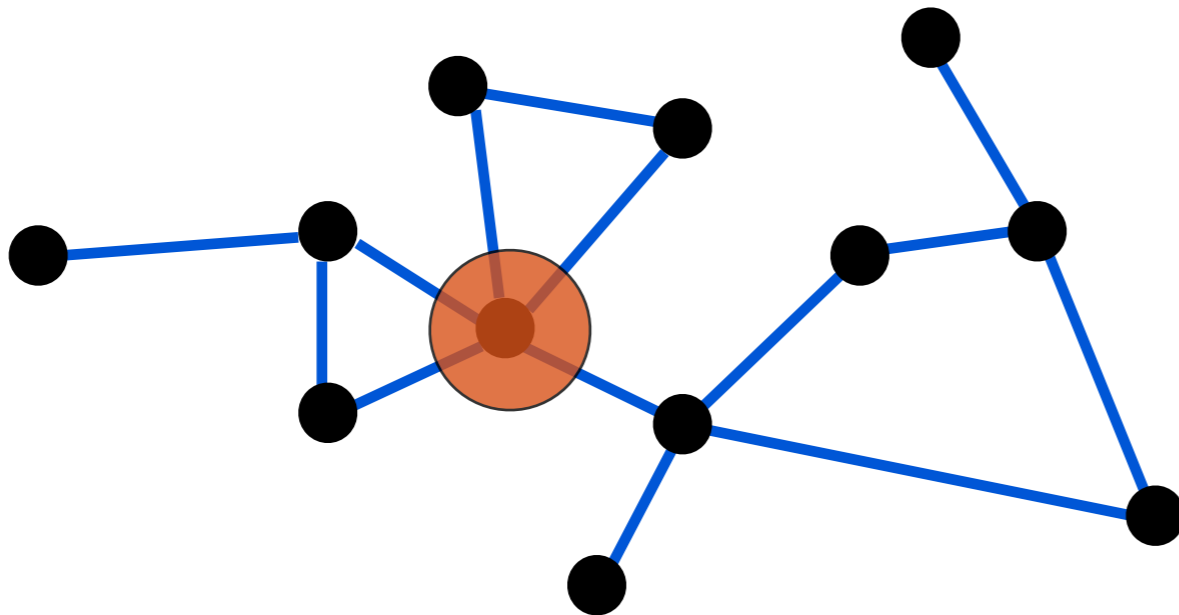
- In general the answer is no!

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L)x$$

- However, if we consider polynomial filters, the answer is yes

**Example:**  $\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \theta \in \mathbb{R}^{K+1} \quad \longrightarrow \quad \hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$

- $L^K$  defines the  $K$ -hop neighborhood:  $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



# Is the graph convolution localized?

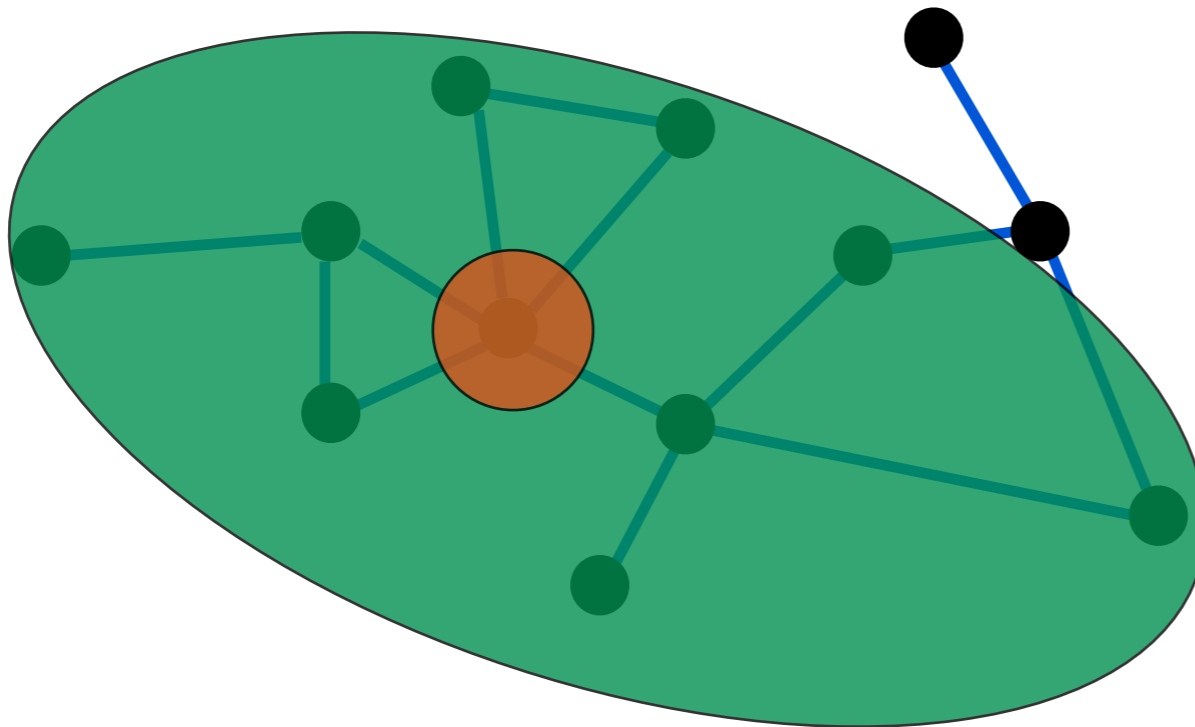
- In general the answer is no!

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L)x$$

- However, if we consider polynomial filters, the answer is yes

**Example:**  $\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \theta \in \mathbb{R}^{K+1} \implies \hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$

- $L^K$  defines the  $K$ -hop neighborhood:  $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



# A spatial interpretation of graph convolution

- Localization of the Laplacian polynomials leads to a spatial interpretation on the graph

$$x * g = \hat{g}(L)x = \sum_{k=0}^K \theta_k L^k x = \sum_{k=0}^K \theta_k z_k$$

- Note that:

$$z_0 = x$$

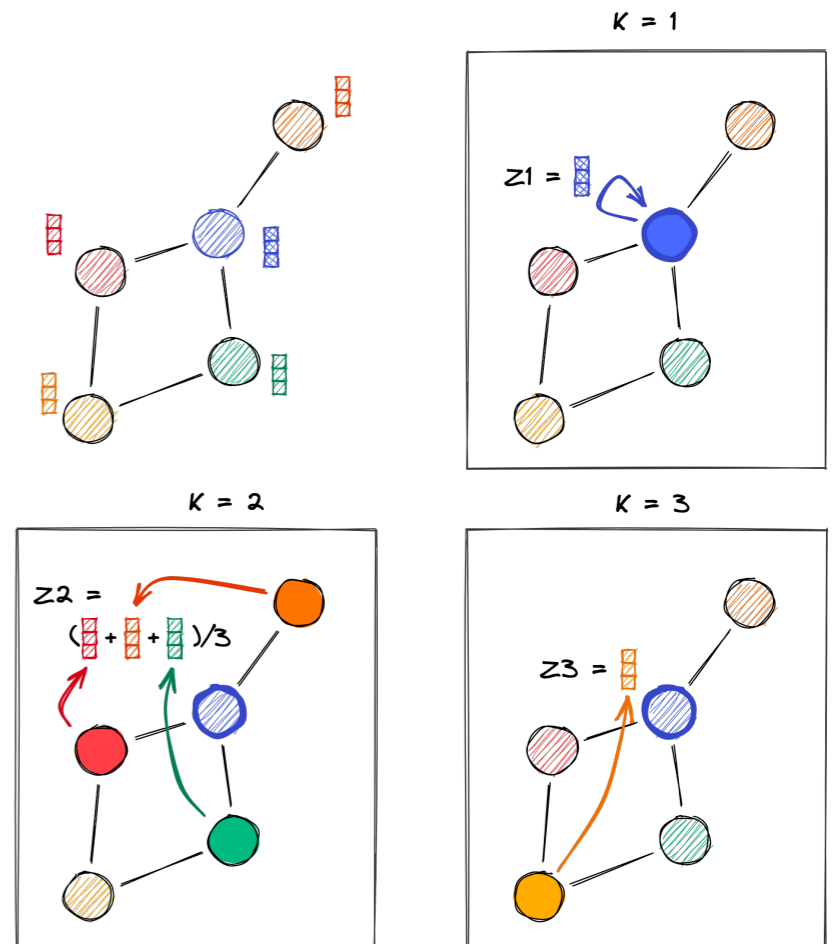
$$z_1 = Lz_0$$

$$z_2 = Lz_1 = L^2 z_0$$

⋮

$$z_K = Lz_{K-1} = \dots = L^K z_0$$

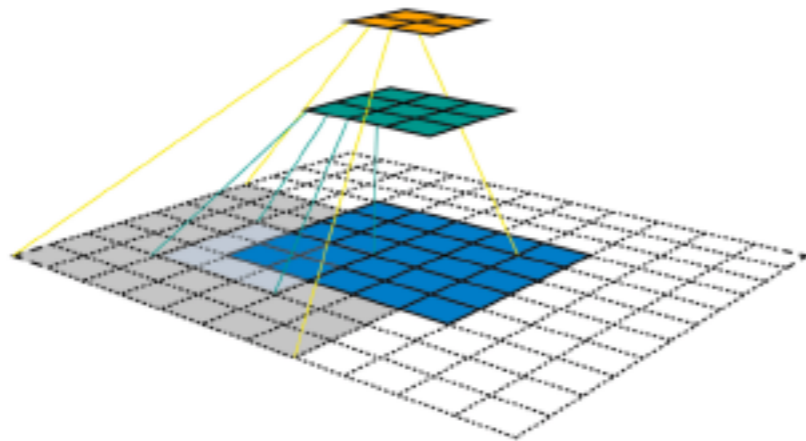
- Graph convolution can be computed recursively by exchanging information in a local neighborhood (i.e., message passing)
- The kernel  $\hat{g}(\cdot)$  does not depend on the order of the nodes: permutation invariant!



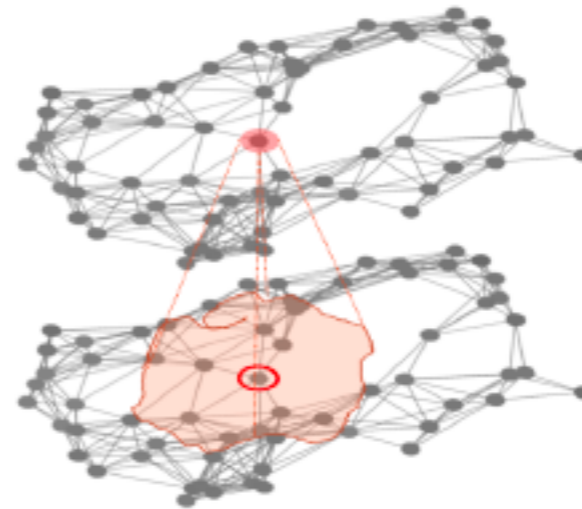
# The receptive field of graph convolution

---

- Node embeddings are based on local neighborhood propagation
- Due to the irregular nature of the graph, there is no fixed size neighbourhood
- The degree  $K$  of the polynomial defines the receptive field of each node



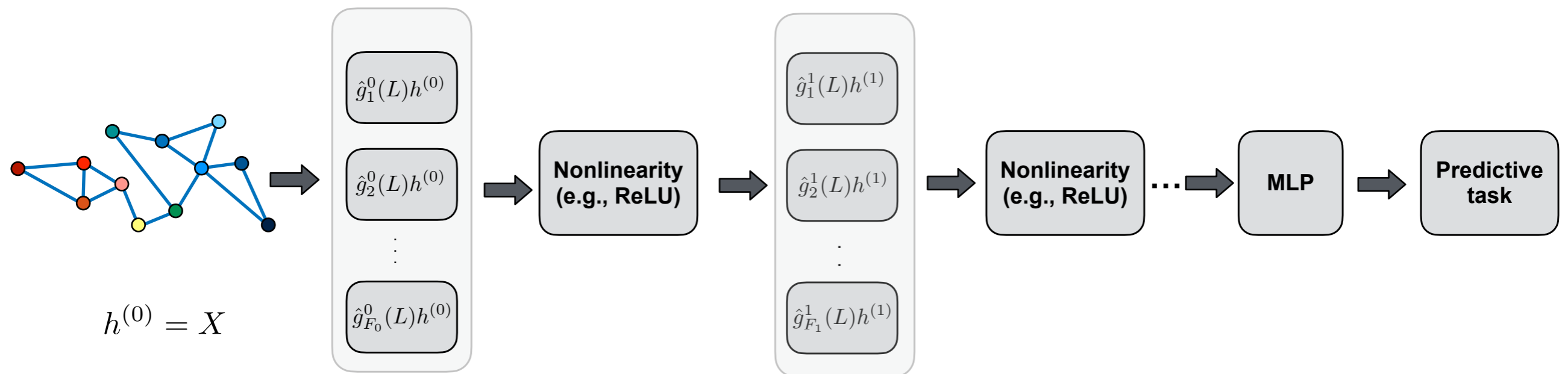
Receptive field on an image



Receptive field on a graph

# The basic GNN: a spectral viewpoint

- Typical GNN architectures consist of a set of graph convolutional layers, each of which is followed by elementwise nonlinearity





- By learning the parameters of the each convolutional filter, we learn how to propagate information on a graph to compute node embeddings


# Spectral approaches in one slide

- Convolution is defined in the graph Fourier domain

$$x *_{\theta} g = \chi \hat{g}(\theta) \chi^T x$$

- **Spectral GCNN:**  $\hat{g}(\lambda) = \theta$    $x * g = \chi \theta \chi^T x$

- **ChebNet:**  $\hat{g}(\lambda) = \sum_{k=0}^K \theta_k T_k(\lambda)$    $x * g = \sum_{k=0}^K \theta_k T_k(L) x$

- **GCN:**  $K = 1$    $x * g = (\theta_0 - \theta_1 D^{-1/2} W D^{-1/2}) x$

- Parameters  $\theta$  are learned through the network

# Graph Convolutional Networks (GCN)

- **Main intuition:** Design a scalable architecture with first-order approximation of spectral graph convolution

$$g * x \approx \theta_0 x + \theta_1 (L - I_N)x = \theta_0 x - \theta_1 D^{-1/2} \boxed{W} D^{-1/2} x$$

$$\Downarrow \quad \theta = \theta_0 = -\theta_1$$

$$g * x \approx \theta (I + D^{-1/2} W D^{-1/2}) x$$

$$\Downarrow \quad \tilde{W} = W + I_N$$

$$g * x \approx \theta \tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2} x$$

Adjacency/Weight matrix

Degree matrix

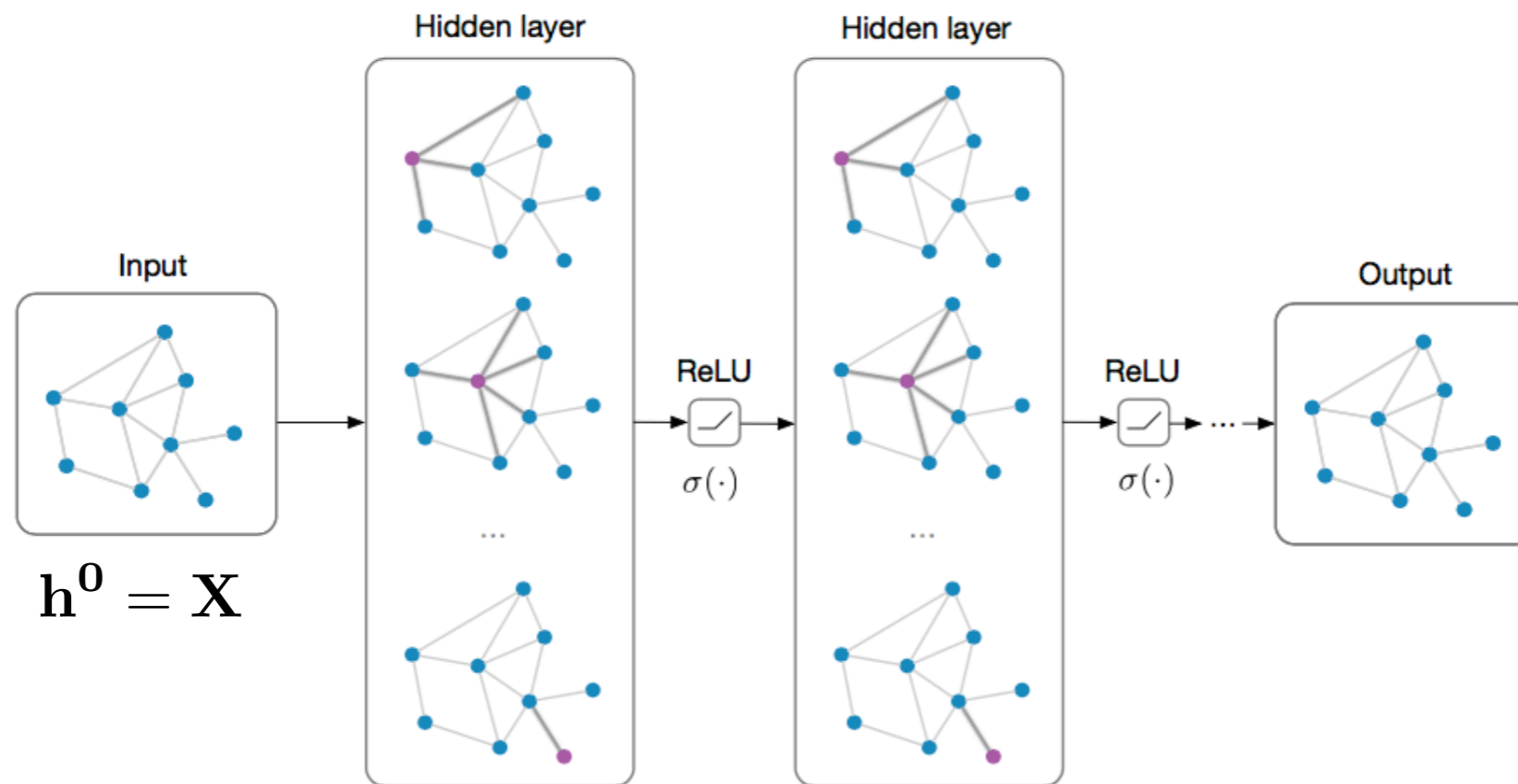
- A convolutional layer is defined as:

$$h^{l+1} = \sigma(\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2} h^l \boxed{\theta^{l+1}})$$

Learned parameters

# GCN architecture

- Very often, it consists of two GCN layers



$$\mathbf{h}^{l+1} = \sigma(\tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{W}} \tilde{\mathbf{D}}^{-1/2} \mathbf{h}^l \theta^{l+1})$$

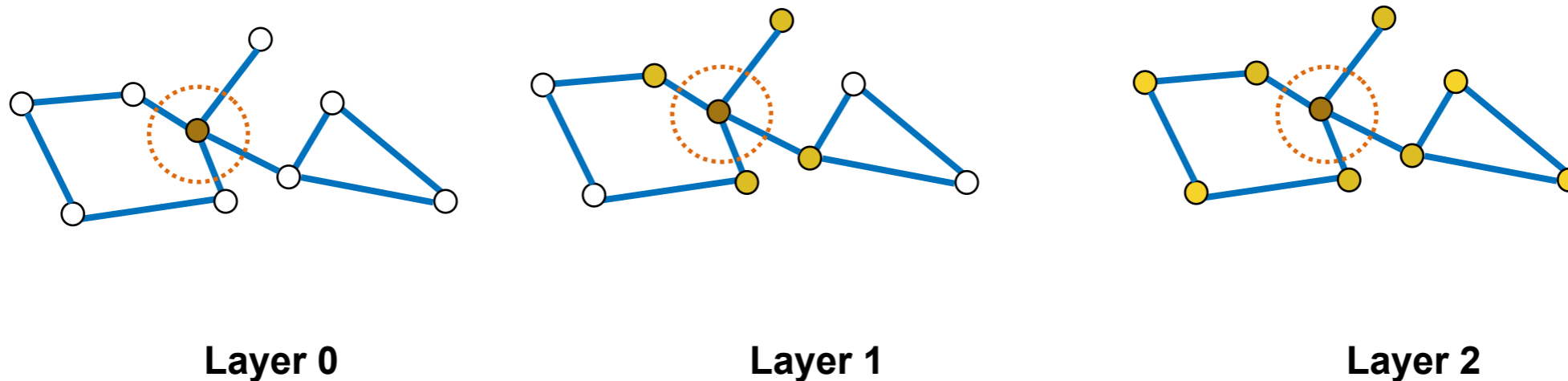
[Kipf et al., Semi-Supervised Classification with Graph Convolutional Networks, ICLR, 2017]



# Each layer increases the receptive field of each node

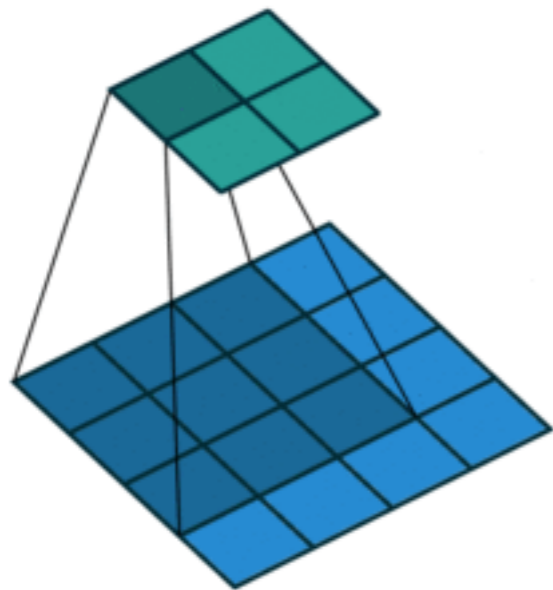
---

- Each layer increases the receptive field by  $K$  hops
- Example:  $K = 1$

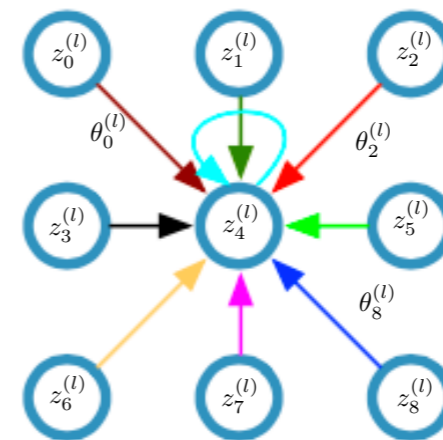


# Towards a graph convolution: A spatial viewpoint

- **Key intuition:** Generalize the notion of convolution from images (grid graph) to networks (irregular graph)
- Example of a single CNN layer with 3x3 filter
  - Fixed neighbourhood
  - Canonical order across neighbors



Animation from V. Dumoulin

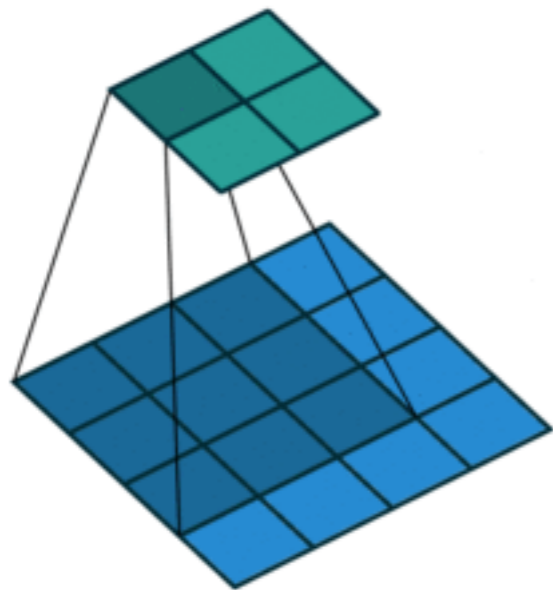


$$z_i^{(l+1)} = \sum_{i=0}^8 \theta_i^{(l)} z_i^{(l)}$$

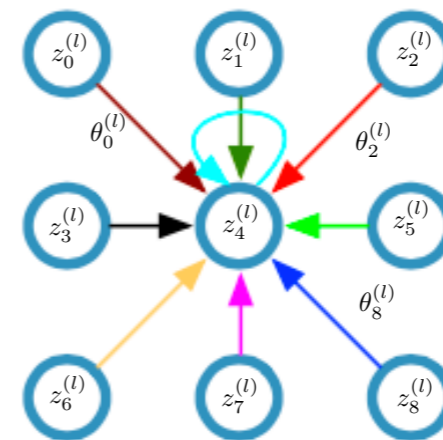
**Can we exploit similar structure for graph data?**

# Towards a graph convolution: A spatial viewpoint

- **Key intuition:** Generalize the notion of convolution from images (grid graph) to networks (irregular graph)
- Example of a single CNN layer with 3x3 filter
  - Fixed neighbourhood
  - Canonical order across neighbors



Animation from V. Dumoulin



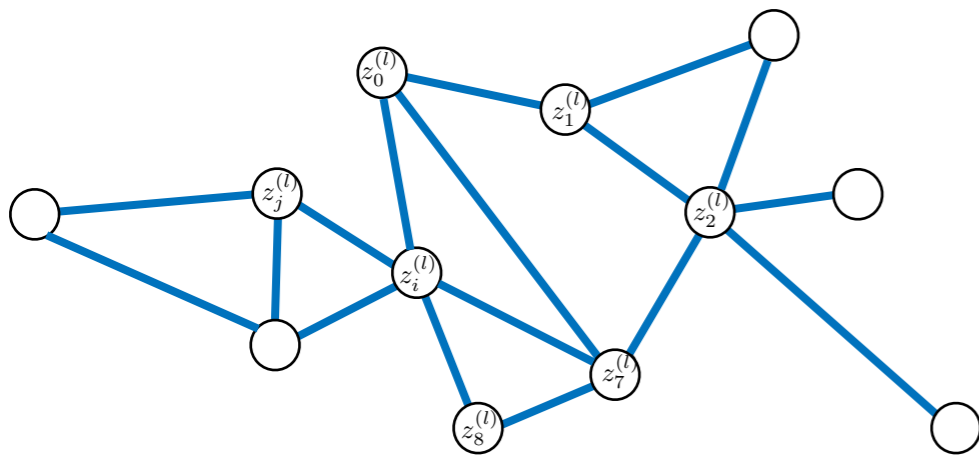
$$z_i^{(l+1)} = \sum_{i=0}^8 \theta_i^{(l)} z_i^{(l)}$$

**Can we exploit similar structure for graph data?**

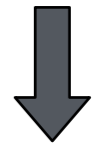
# Spatial graph convolution

- Main issue: We cannot have variable number of weights; it requires assuming an order on the nodes

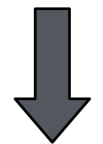
- Solution: Impose same filter weights for all nodes



$$z_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \theta_j^{(l)} z_j^{(l)}$$



$$z_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \theta^{(l)} z_j^{(l)}$$

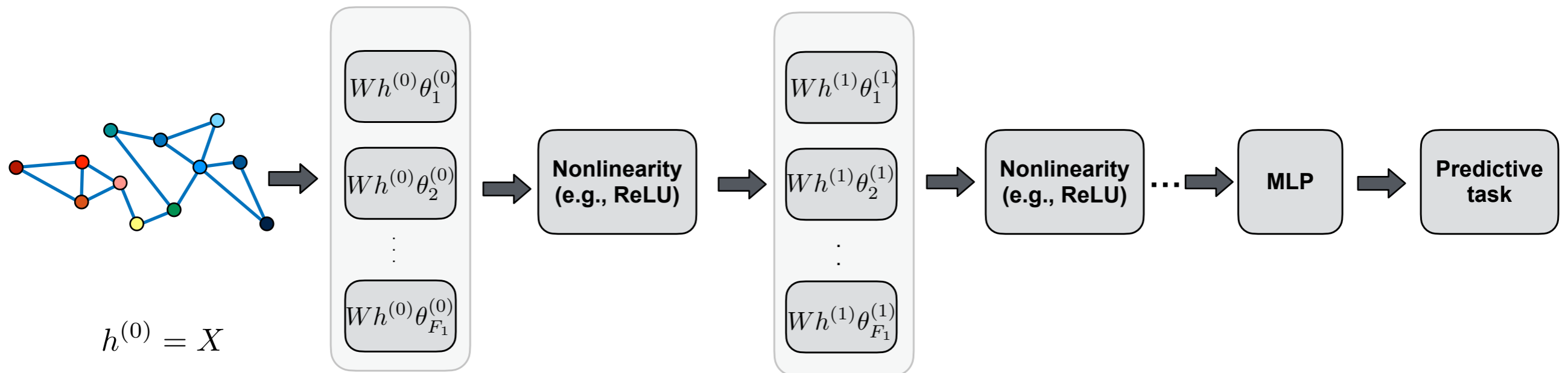


$$z_i^{(l+1)} = \theta^{(l)} z_i^{(l)} + \sum_{j \in \mathcal{N}_i} \theta^{(l)} z_j^{(l)}$$

Update embeddings by exchanging information with 1-hop neighbors

# The basic GNN: a spatial viewpoint

- Consists of a set of graph convolutional layers, each of which is followed by elementwise nonlinearity, i.e.,  $h^{(l+1)} = \sigma(z^{(l)})$



- Each layer increases the receptive field by 1-hop neighbors

# Message Passing Neural Network (MPNN)

---

- **Main intuition:** Each node exchange messages with its neighbors and update its representations based on these messages
- The message passing scheme runs for T time steps and updates the representation of each vertex based on its previous representation and the representation of its neighbors

$$m_i^{l+1} = \sum_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l, e_{ij})$$

Message function

$$h_i^{l+1} = U_l(h_i^l, m_i^{l+1})$$

Vertex update function

**Learned differentiable functions!**

[Gilmer et al, Neural Message Passing for Quantum Chemistry, ICML, 2017]

# MPNN - Example

---

- At each iteration, the embeddings are updated as follows:

$$h_1^{l+1} = \theta_0^l h_1^l + \theta_1^l h_2^l + \theta_1^l h_3^l$$

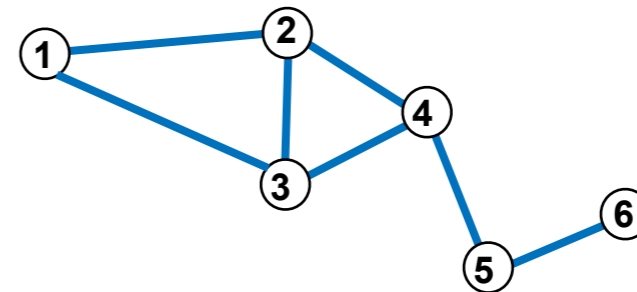
$$h_2^{l+1} = \theta_0^l h_2^l + \theta_1^l h_1^l + \theta_1^l h_3^l + \theta_1^l h_4^l$$

$$h_3^{l+1} = \theta_0^l h_3^l + \theta_1^l h_1^l + \theta_1^l h_2^l + \theta_1^l h_4^l$$

$$h_4^{l+1} = \theta_0^l h_4^l + \theta_1^l h_2^l + \theta_1^l h_3^l + \theta_1^l h_5^l$$

$$h_5^{l+1} = \theta_0^l h_5^l + \theta_1^l h_4^l + \theta_1^l h_6^l$$

$$h_6^{l+1} = \theta_0^l h_6^l + \theta_1^l h_5^l$$



- The output of the message passing is:

$$\{h_1^{l_{max}}, h_2^{l_{max}}, h_3^{l_{max}}, h_4^{l_{max}}, h_5^{l_{max}}, h_6^{l_{max}}\}$$

# Comparison between spatial and spectral design

---

- Spectral convolution: Generalizes the notion of convolution by following a frequency viewpoint
- Spatial convolution: Generalizes the notion of convolution by following a spatial viewpoint
- Strong links exist between both; The practical difference usually relies on the receptive field
  - Spectral approaches: Every layer can 'reach' K-hops neighbors
  - Spatial approaches: Each layer can 'reach' 1-hops neighbors



# A summary of the GNN landscape

---

- **Convolutional GNNs** can be generalised as:

$$h_i = \phi\left(X_i, \bigoplus_{j \in \mathcal{N}_i} \psi(X_j)\right)$$

- **Message passing GNNs:**

$$h_i = \phi\left(X_i, \bigoplus_{j \in \mathcal{N}_i} \psi(X_i, X_j)\right)$$

- **Attentional GNNs:**

$$h_i = \phi\left(X_i, \bigoplus_{j \in \mathcal{N}_i} \alpha(X_i, X_j) \psi(X_j)\right)$$

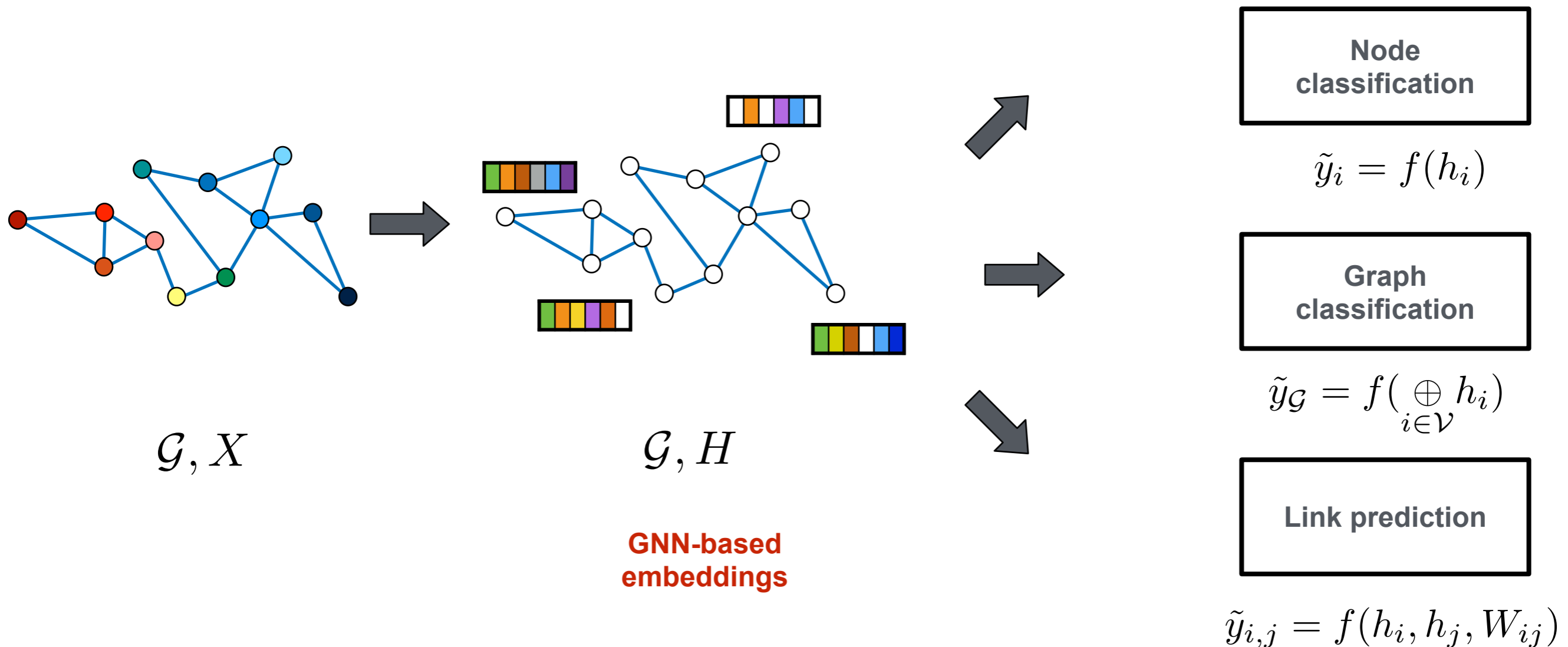
**Functions to be learned!**

- Depending on how these functions are instantiated, different architectures are obtained

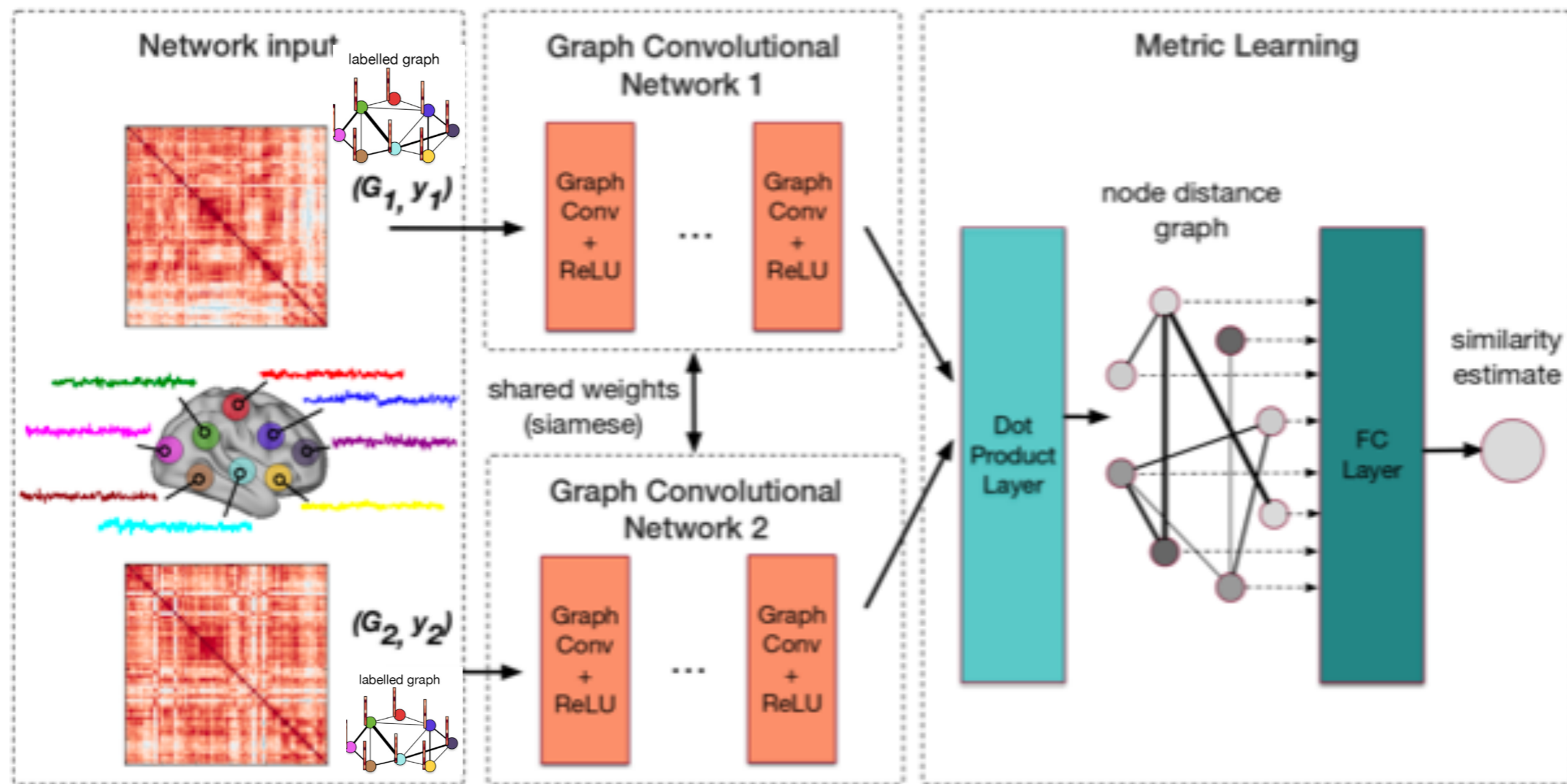
[Slide inspired from P. Veličković]

# How to use GNNs?

- GNNs typically provide embeddings at a node level
- These embeddings can be used for learning a downstream task



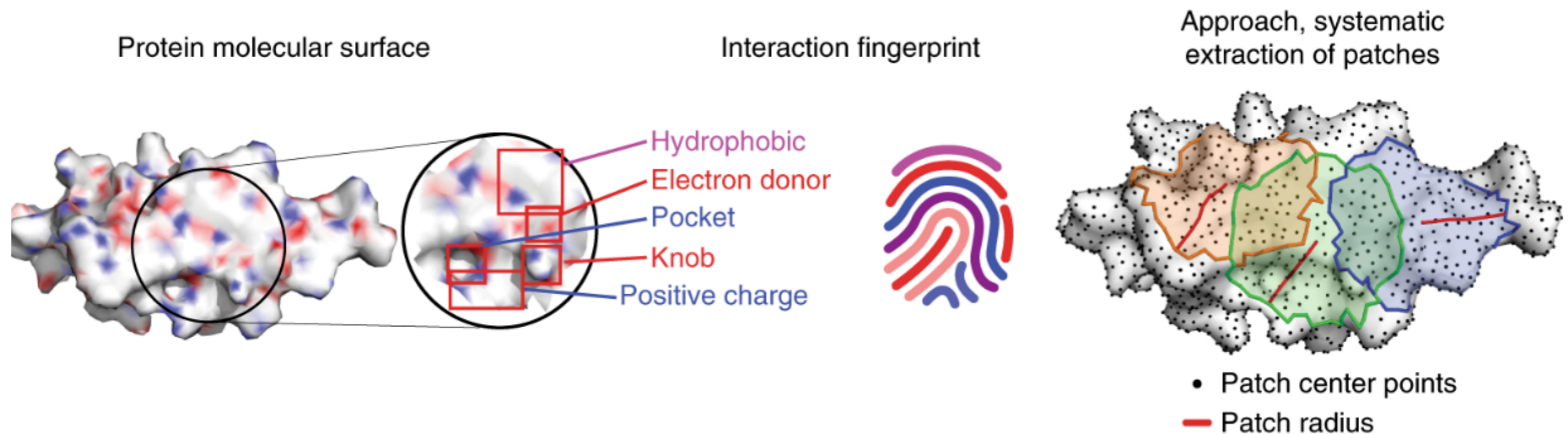
# Neuroscience: learn to compare brain networks



[Ktena et al., Metric learning with spectral graph convolutions on brain connectivity networks, NeuroImage, 2018]

# Protein-protein interactions

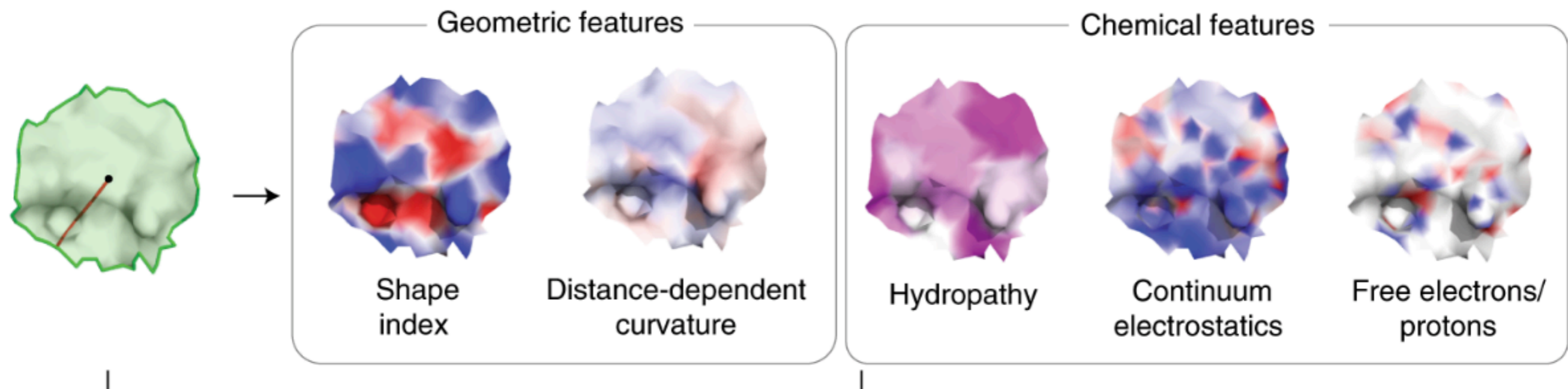
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

# Protein-protein interactions

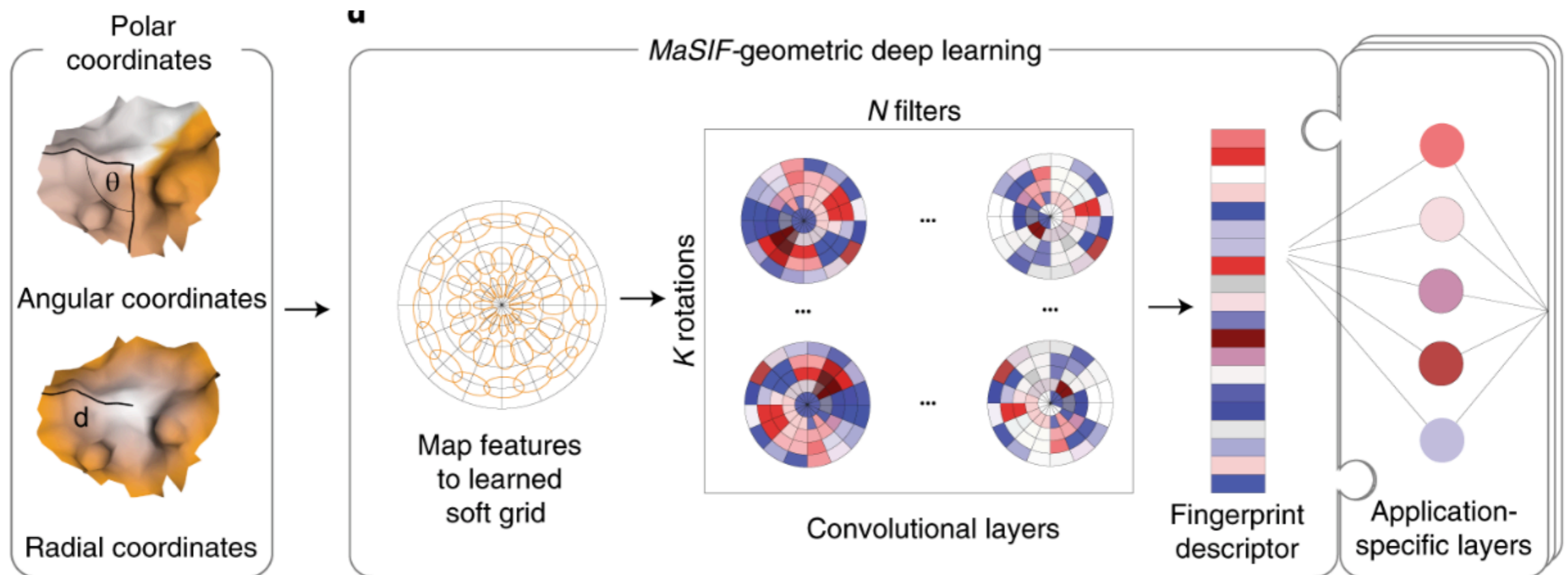
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

# Protein-protein interactions

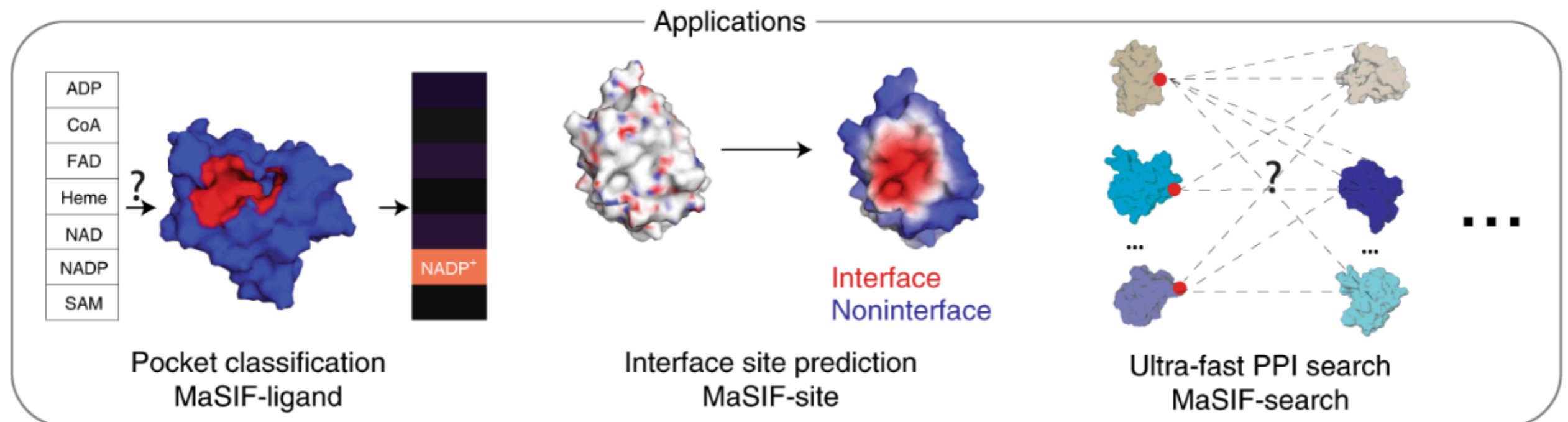
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

# Protein-protein interactions

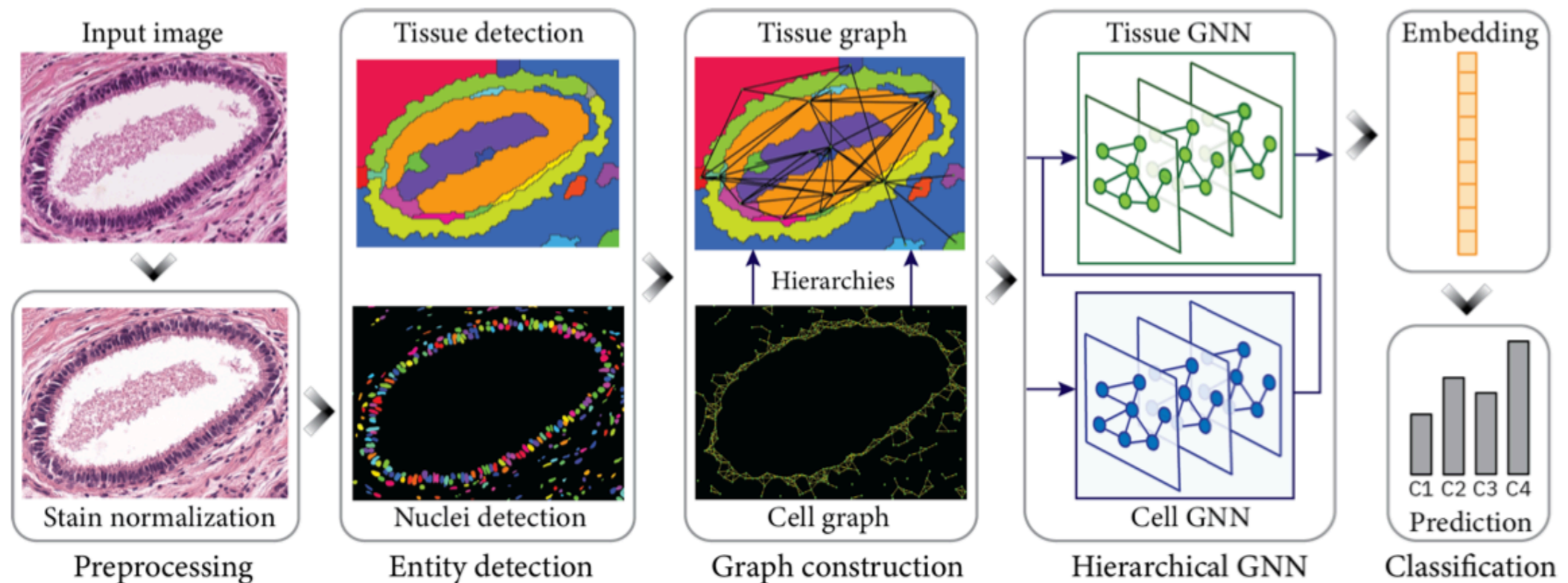
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

# Medical imaging

- Digital pathology: Graph based representations provide a flexible tool for modelling complex dependencies at different levels of hierarchy (e.g., cells, tissues)



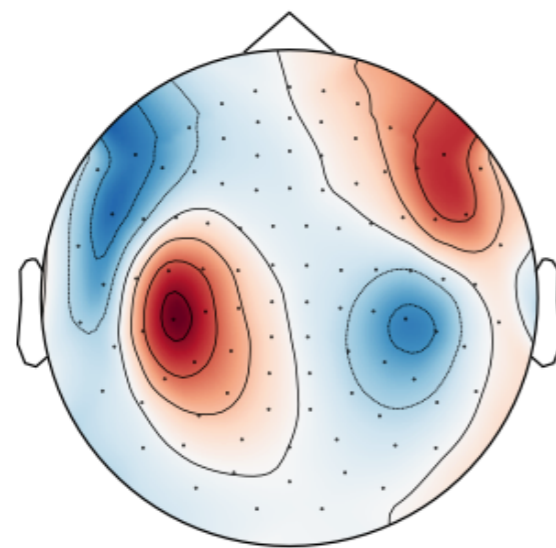
[Pati et al, "Hierarchical graph representation in digital pathology," MEDIA, 2022]

[Li et al, Representation learning for networks in biology and medicine: Advancements, challenges, and opportunities, arXiv, 2021]

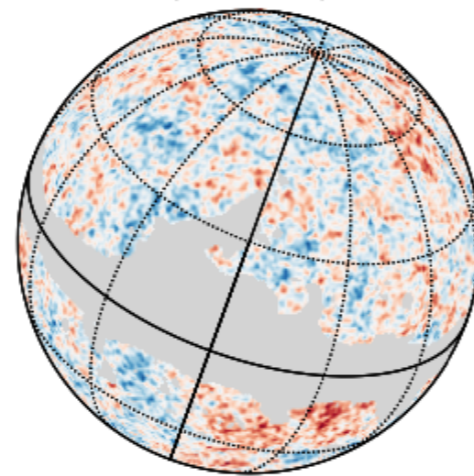


# Spherical imaging

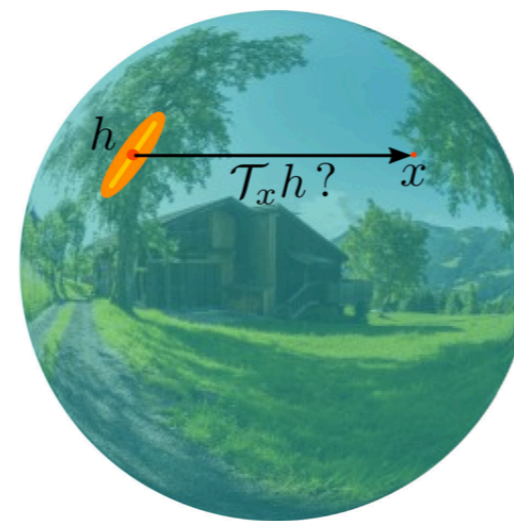
- Spherical data has specific spatial and statistical properties that cannot be captured by regular CNN models



Brain activity (MEG)



Cosmic microwave background temperature



Omnidirectional images

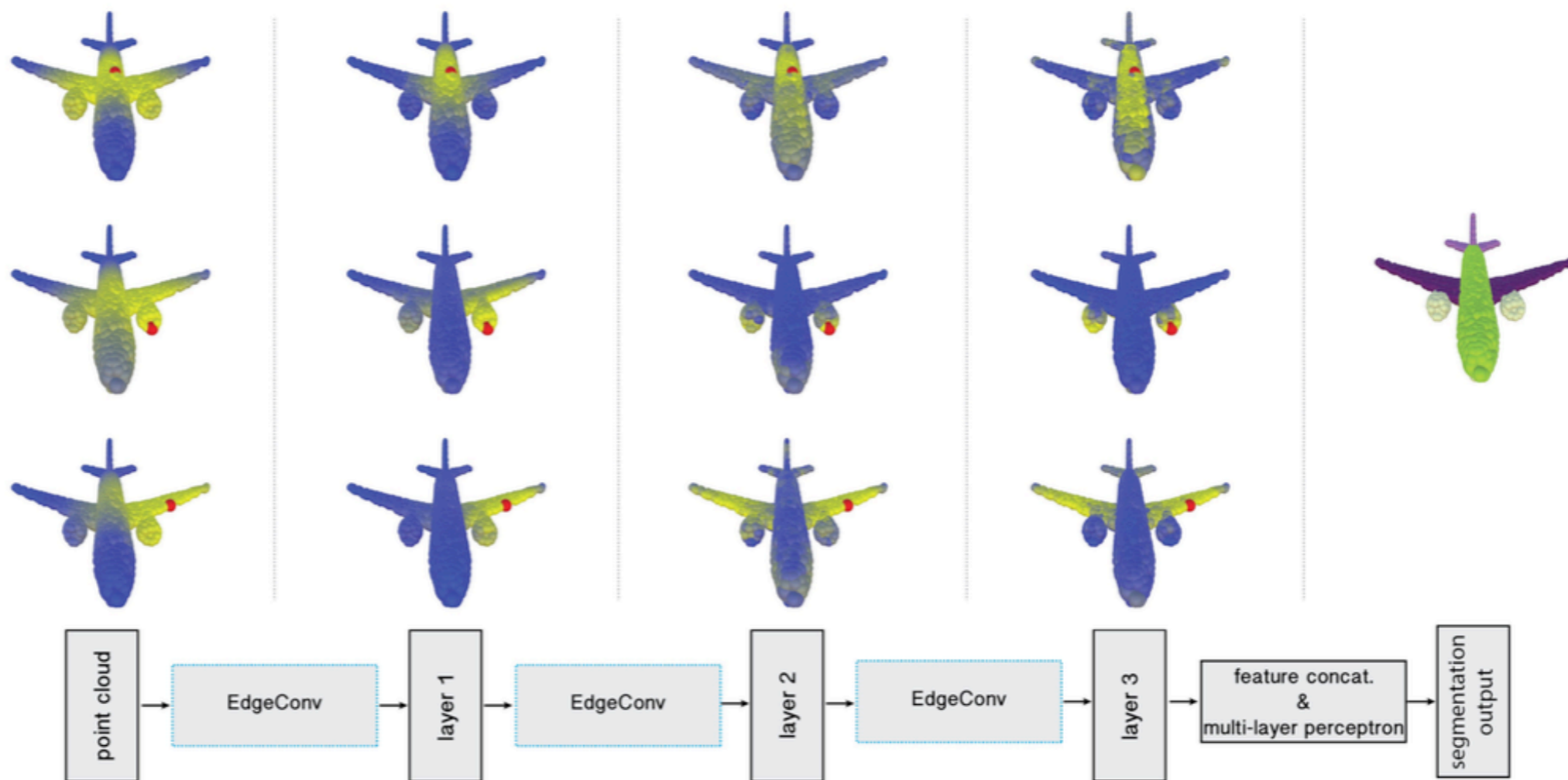
- Sphere is modelled as a graph and classical operation (convolution, translation, pooling...) are performed on the graph

[Perraudin et al., "DeepSphere", Astronomy and Computing, 2019]

[Bidgoli et al, OSLO: On-the-Sphere Learning for Omnidirectional images and its application to 360-degree image compression, arXiv, 2021]

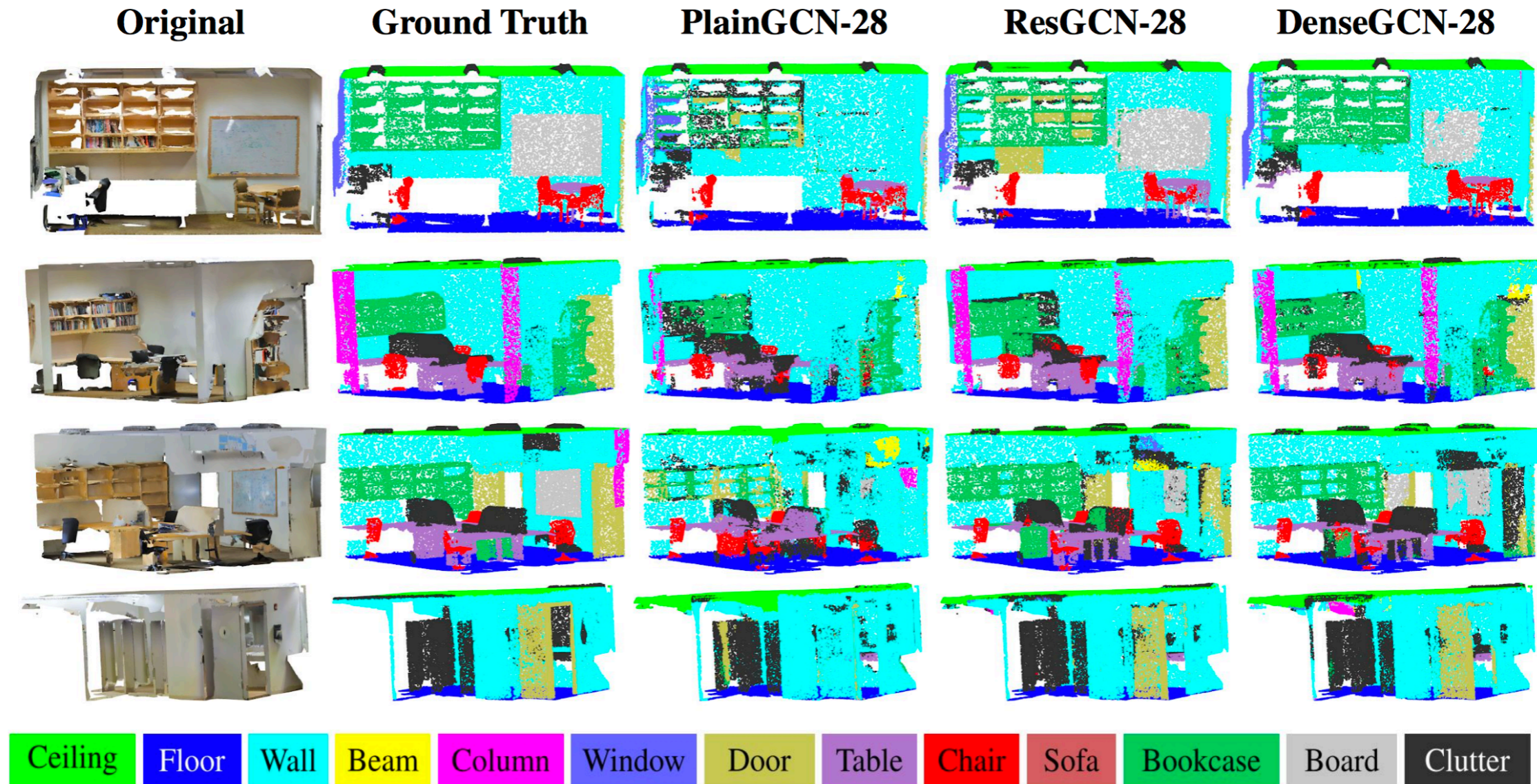
# Point cloud semantic segmentation

- Graph attention convolution are successful in capturing specific shapes that adapt to the structure of an object



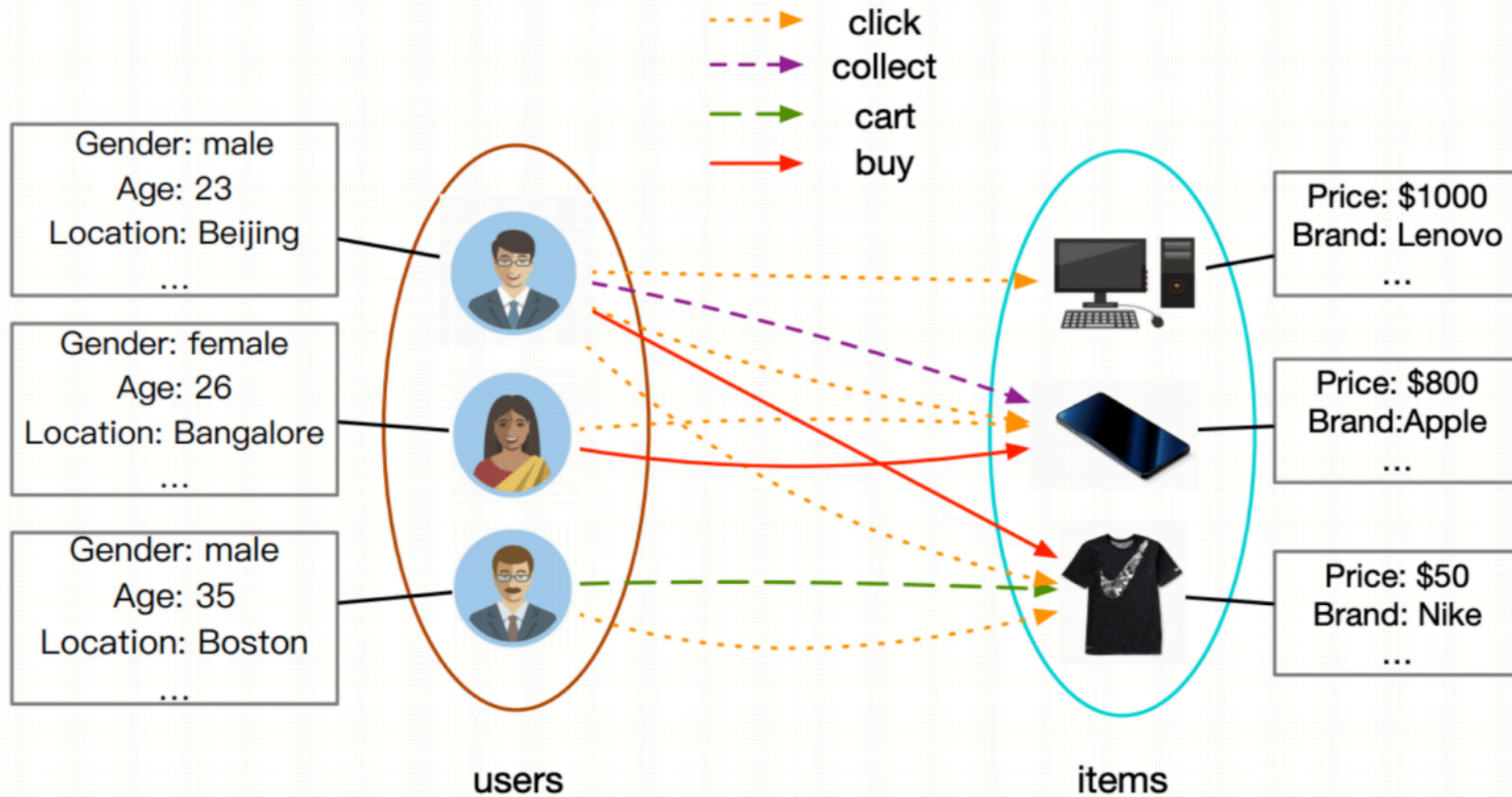
[Wang et al., Graph Attention Convolution for Point Cloud Semantic Segmentation, CVPR, 2019]

# Point clouds semantic segmentation



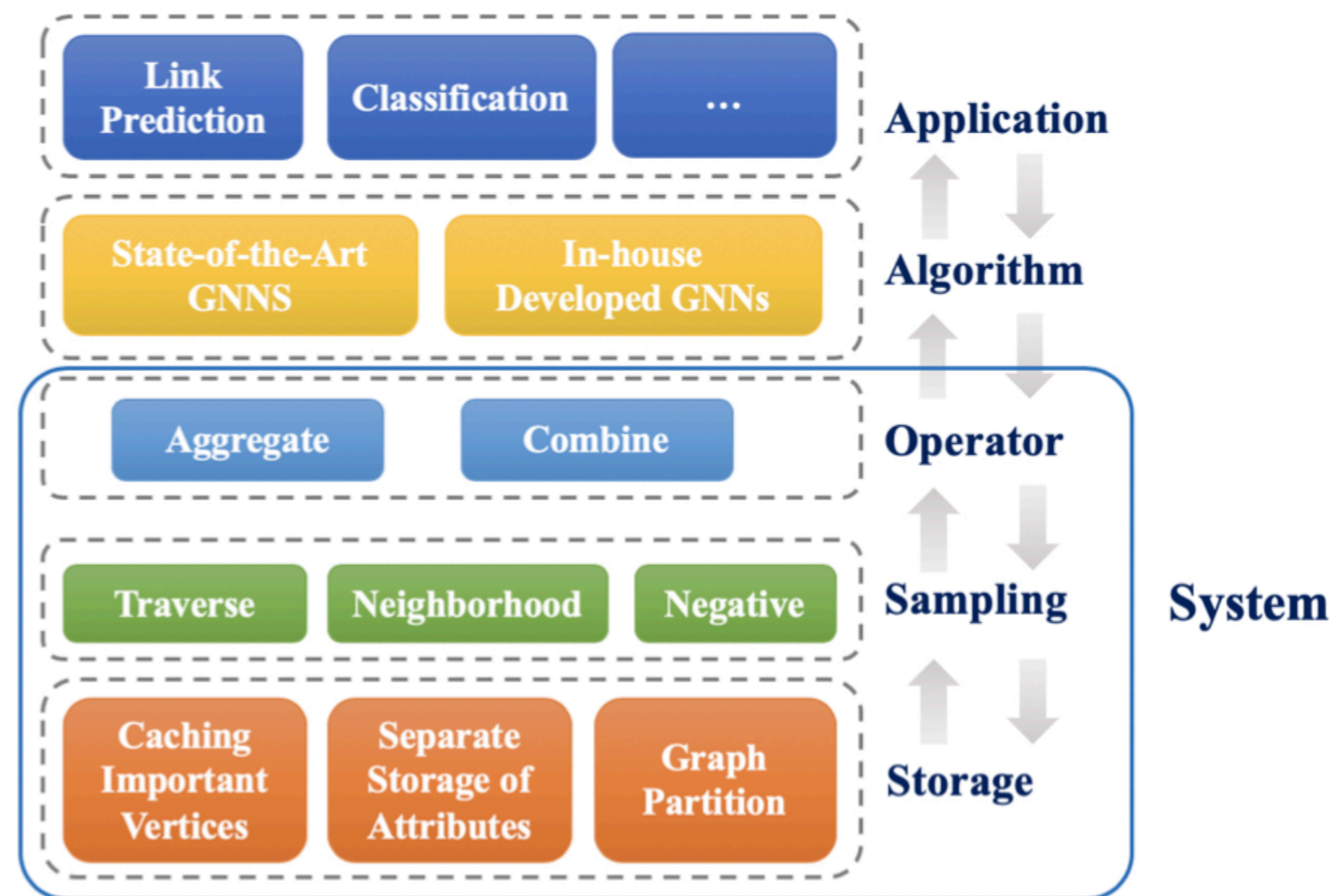
[Li et al., DeepGCNs: Can GCNs Go as Deep as CNNs?, ICCV 2019]

# Recommender systems



# Recommender systems: Aligraph

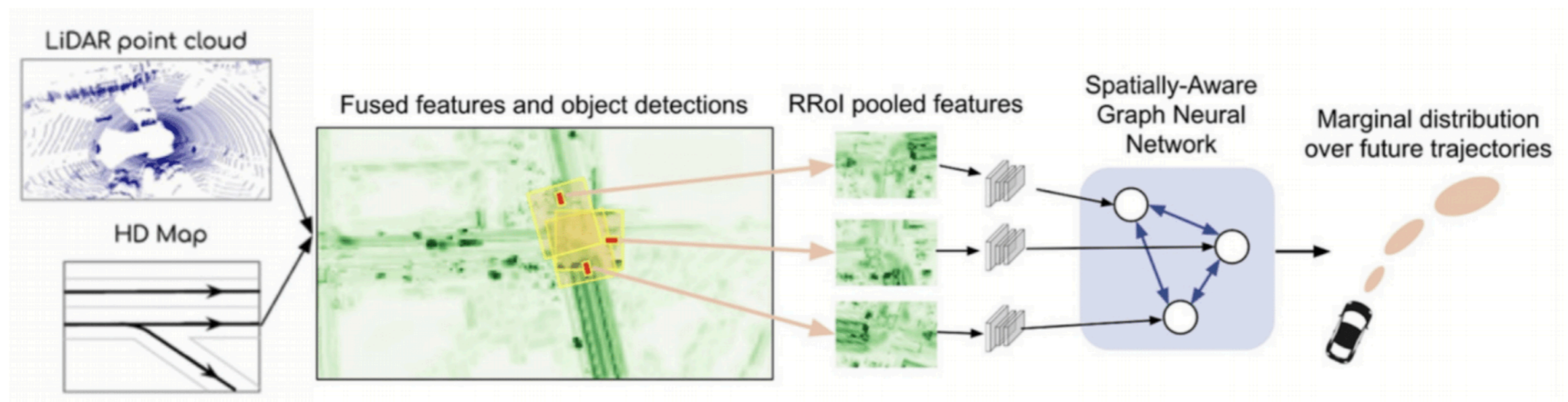
- The system is currently deployed at Alibaba to support product recommendation and personalized search at Alibaba's E-Commerce platform



[Zhu et al., *AliGraph: A Comprehensive Graph Neural Network Platform*, 2019]

# Self driving cars

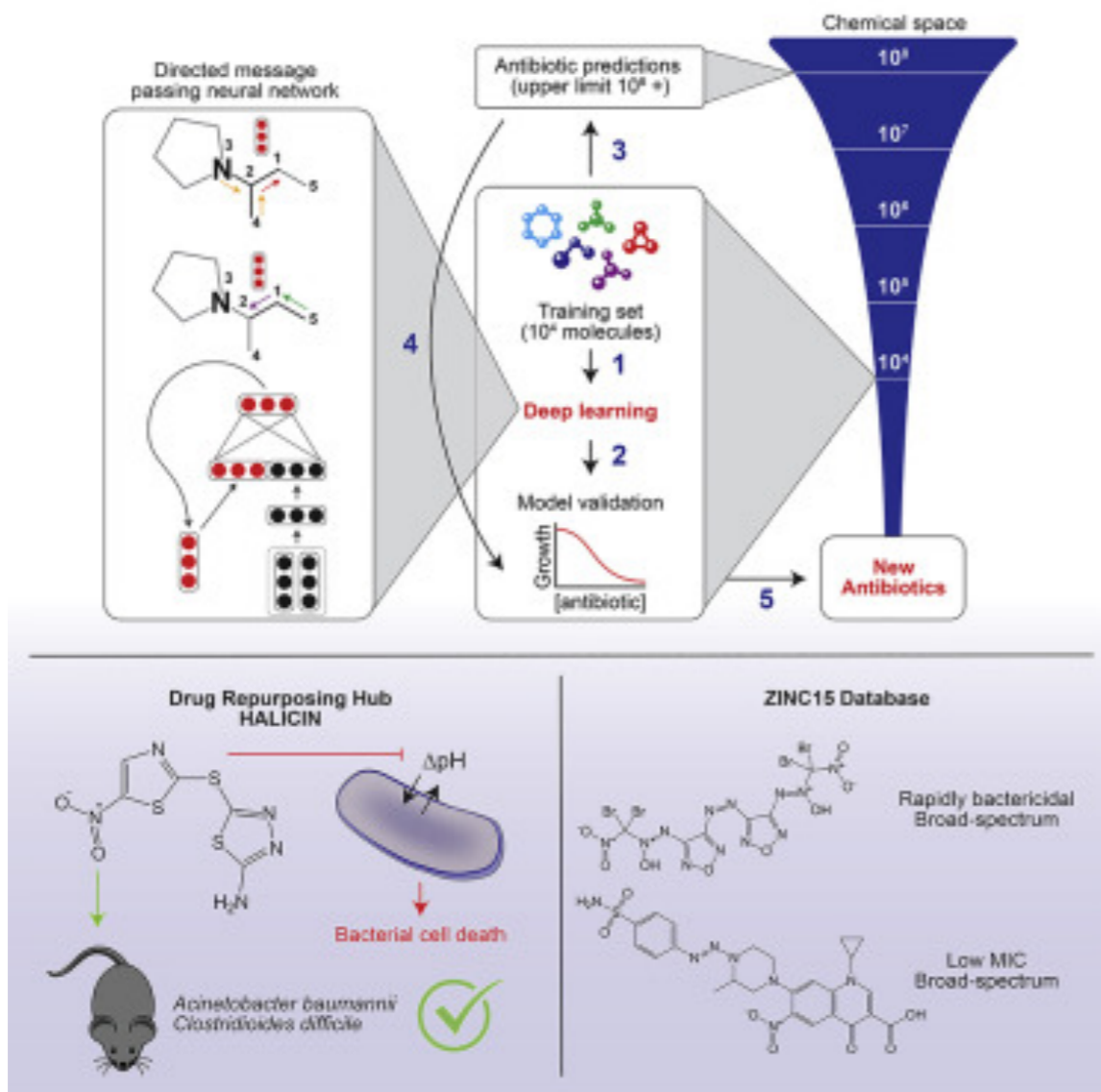
- GNNs provide probabilistic estimates of future trajectories
  - A CNN detects objects
  - A GNN captures interactions between objects and predicts behaviors



[Casas et al, Spatially aware graph neural networks for relational behaviour forecasting for sensor data, ICRA, 2020]

# Molecular graph generation

- Recent advances in antibiotic discovery ...

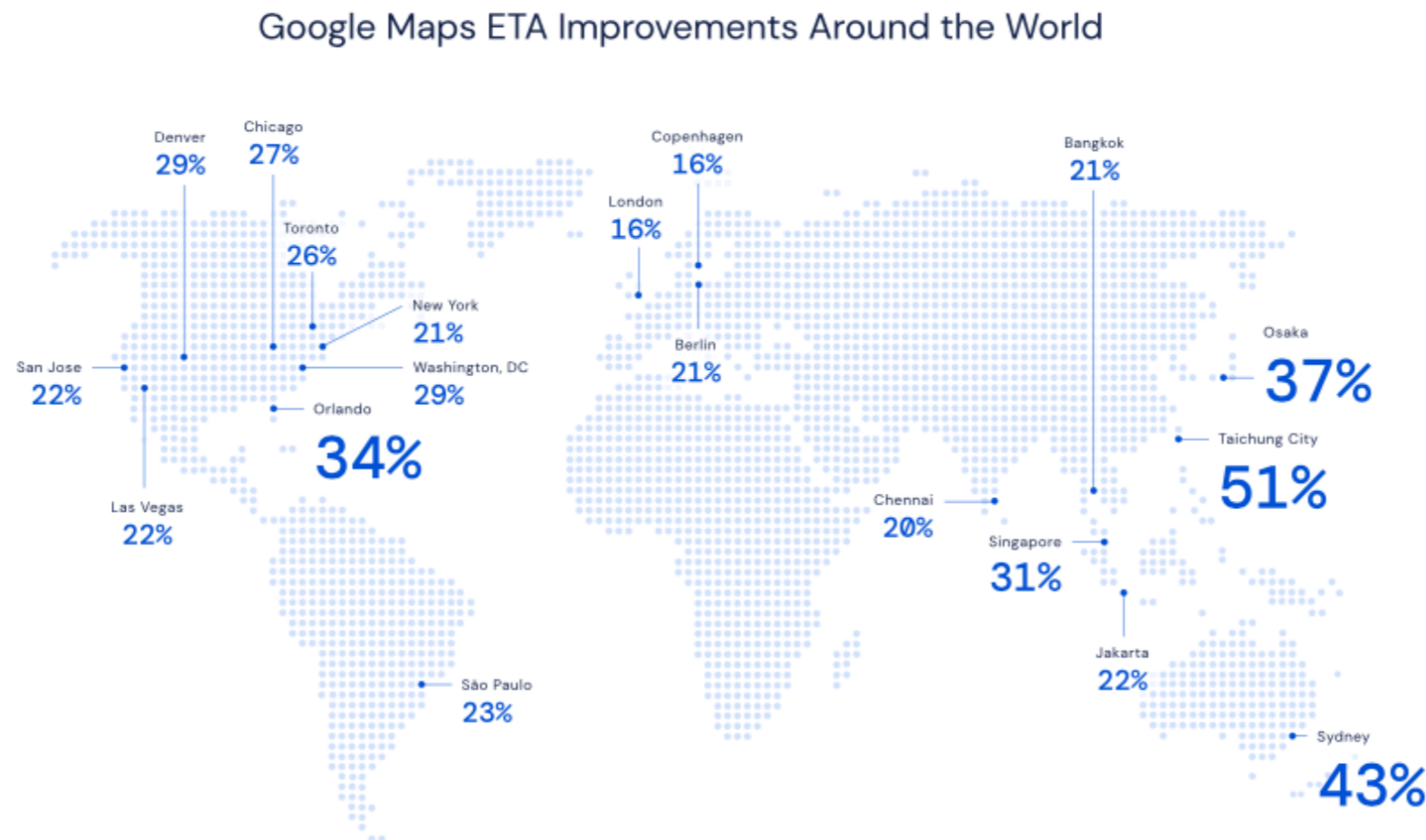


The screenshot shows a news article from the Guardian. The headline is "Scientists discover powerful antibiotic using AI", dated 21 February 2020. The article is categorized under "Antibiotics" and "Health". A sub-headline reads "Do we ask too much of our planet?". The article text includes "Team at MIT says halicin kills some of the world's most dangerous strains". The author is identified as Ian Sample, Science editor. The article is noted as being more than 1 year old. The image shows a person in a lab coat looking through a microscope.

[Simonovsky et al, 2017, De Cao et al 2018, Stokes et al 2020]

# Traffic prediction

- As the road network is naturally modelled by a graph of road segments and intersections, ETA prediction can be improved with graph representation learning

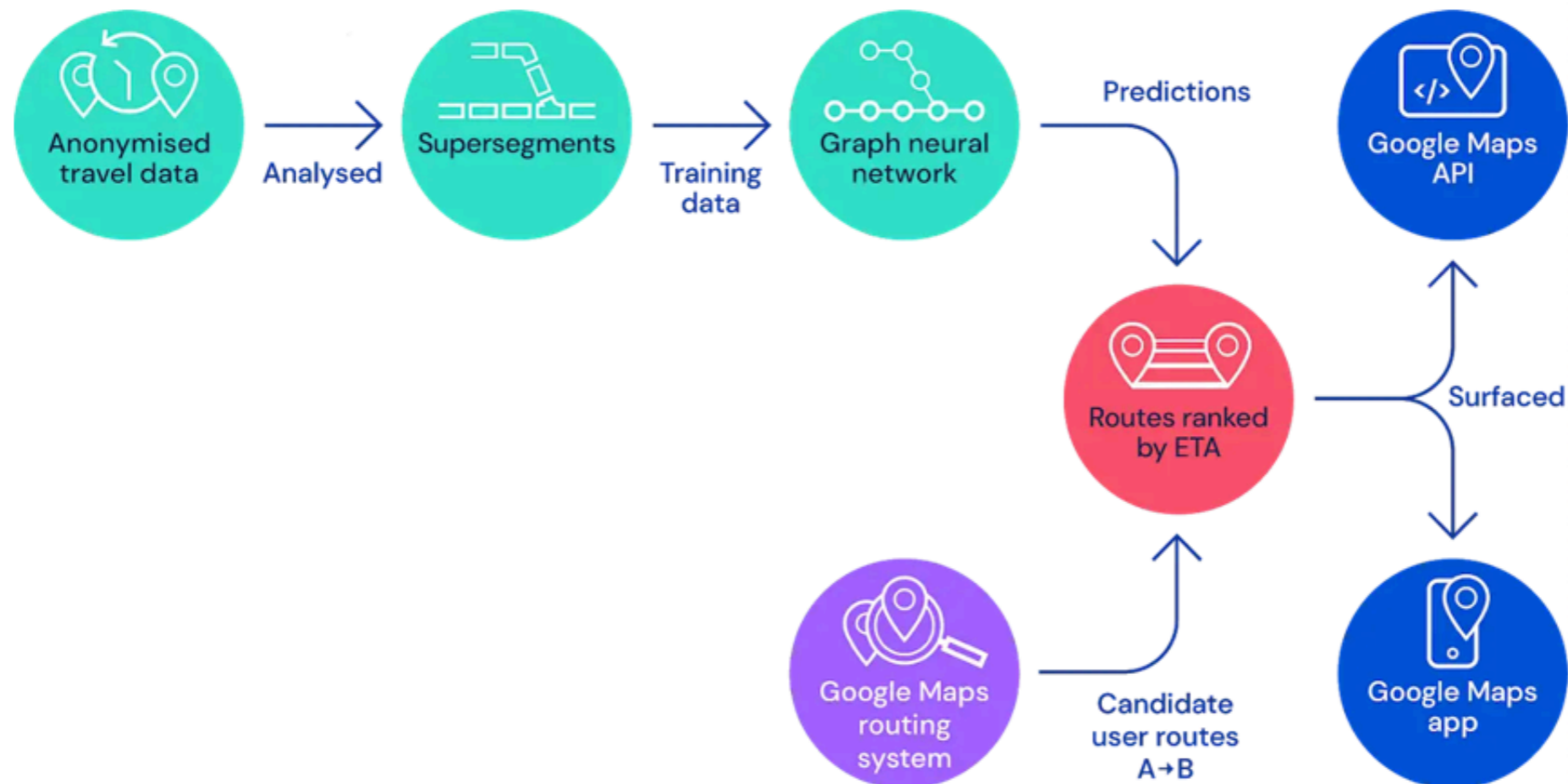


[Derrow-Pinion et al., ETA Prediction with Graph Neural Networks in Google Maps, 2021]



# Traffic prediction

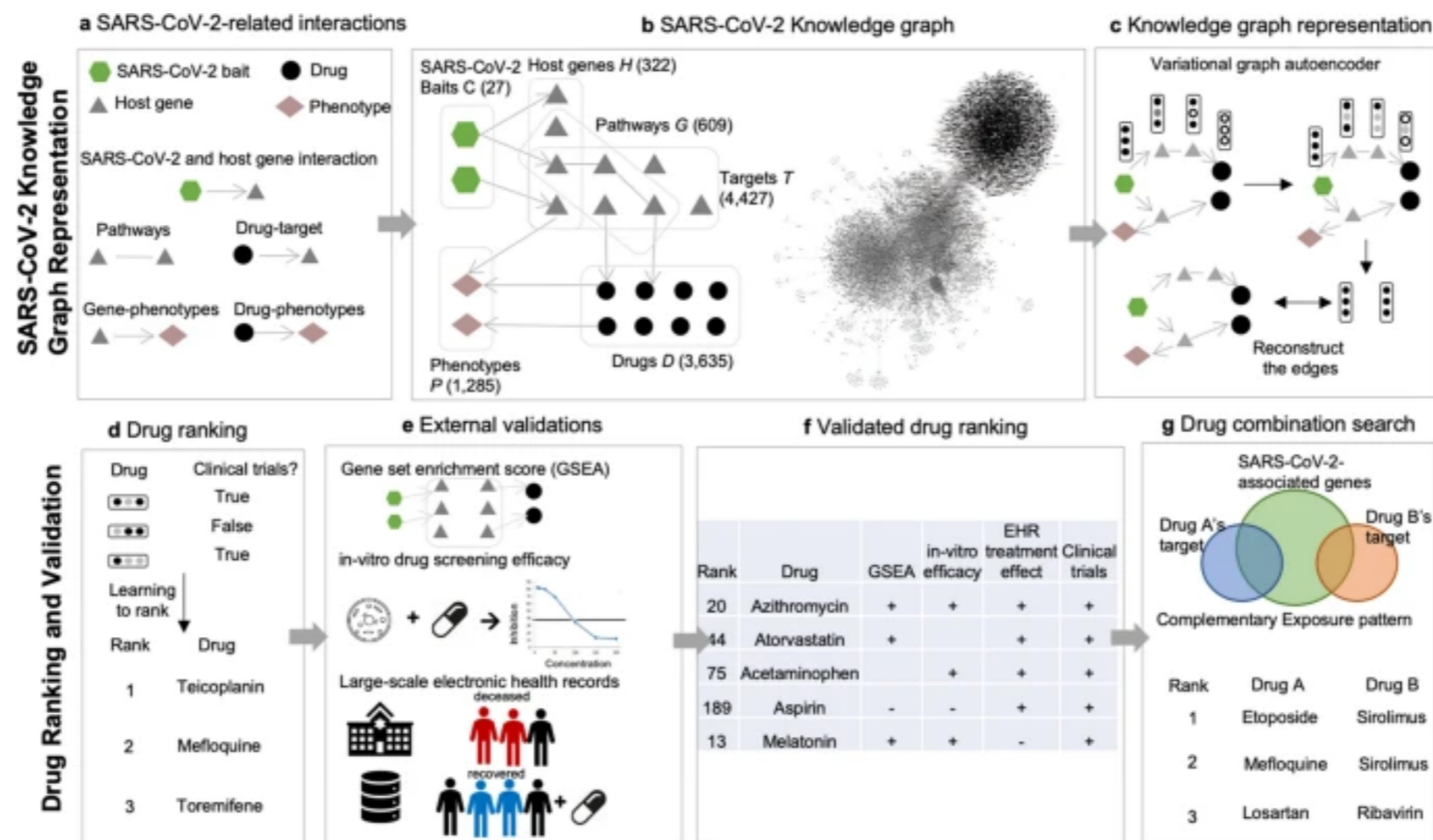
- As the road network is naturally modelled by a graph of road segments and intersections, ETA prediction can be improved with graph representation learning



[Derrow-Pinion et al., ETA Prediction with Graph Neural Networks in Google Maps, 2021]

# Drug repurposing for COVID-19

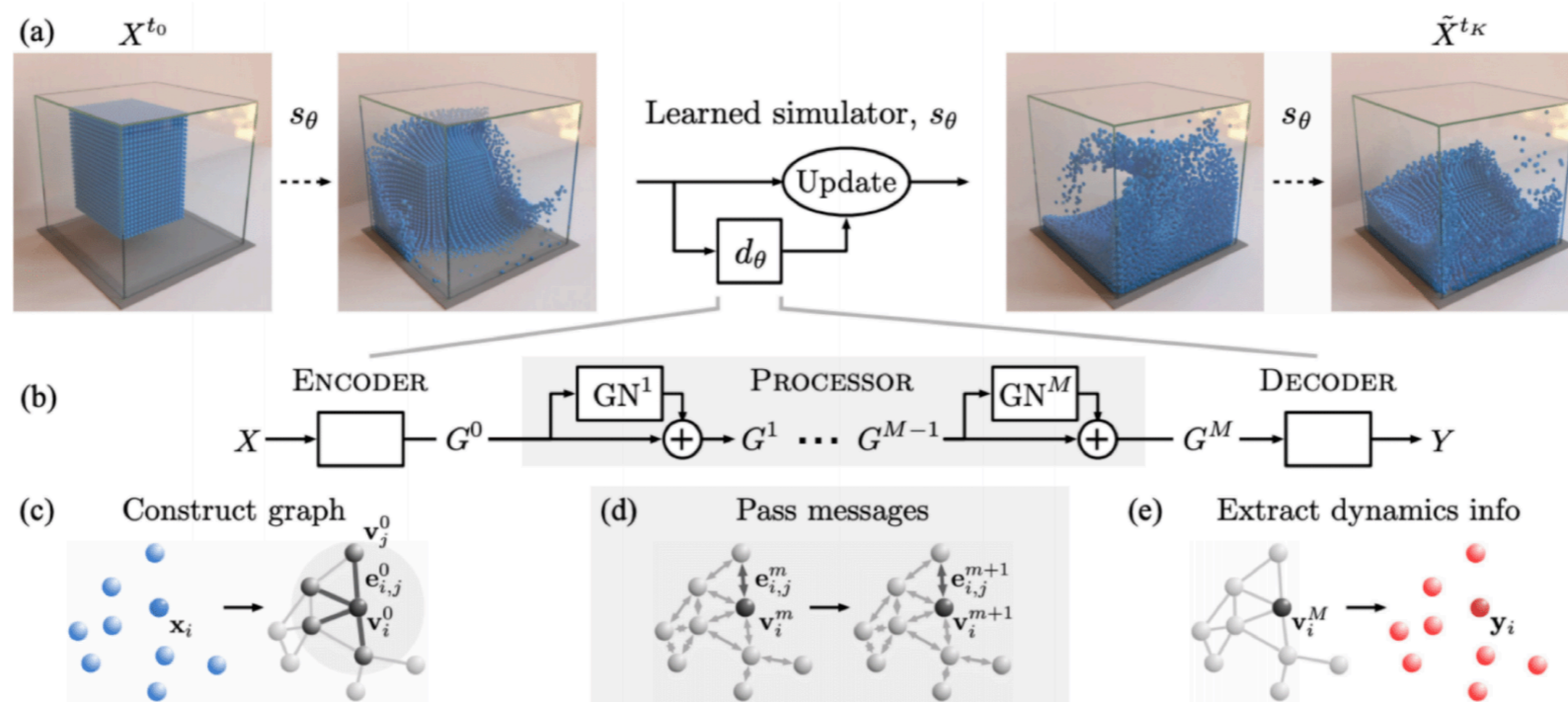
- Deep GNN approaches have been used to derive the candidate drug's representation based on the biological interactions



[Hsieh et al, Drug repurposing for COVID-19 using graph neural network and harmonizing multiple evidence, Nature Sc. Rep., 2021]

# Learning physical simulations

- Mesh-based simulations are central to modeling complex physical systems
- High-dimensional scientific simulations are very expensive
- GNNs have been used to learn mesh-based simulations and predict the dynamics of physical systems



[Sanchez-Gonzales et al, Learning to Simulate Complex Physics with Graph Networks, ICML 2020]

[Pfaff et al, Learning mesh-based simulation with Graph Networks, ICML 2021]

# Useful resources

---

- **Toolboxes**

- [https://github.com/rusty1s/pytorch\\_geometric](https://github.com/rusty1s/pytorch_geometric)
- <https://github.com/dmlc/dgl>
- <https://github.com/deepmind/jraph>
- <https://github.com/tensorflow/gnn>

- **Datasets**

- All GNN libraries contain different datasets
- <https://chrsmrrs.github.io/datasets/>
- <https://chrsmrrs.github.io/datasets/>
- <https://ogb.stanford.edu>

# Some interesting links

---

<https://towardsdatascience.com/graph-ml-in-2022-where-are-we-now-f7f8242599e0#2ddd>

<https://towardsdatascience.com/predictions-and-hopes-for-geometric-graph-ml-in-2022-aa3b8b79f5cc>

# Summary

---

- Machine learning on graphs/networks requires developing new tools that extract information (i.e., features) from complex structures
- Graph-based features (i.e., embeddings) can be designed based on some prior, or learned from data
- Graph neural networks: A very active area of research
  - Different architecture designs, most of them can be categorized as convolutional, message passing, attentional
- A variety of applications, and probably many more to come!

# References

---

1. Graph representation learning, William Hamilton
2. Metrics for graph comparison: A practitioner's guide, Wills, PLOS ONE, 2020
  - <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0228728>
3. Graph Kernels State-of-the-Art and Future Challenges, Borgwardt et al,
  - <https://arxiv.org/pdf/2011.03854.pdf>
4. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, Bronstein et al, 2022
  - <https://arxiv.org/abs/2104.13478>
5. A Comprehensive Survey on Graph Neural Networks, Wu et al, 2021
  - <https://arxiv.org/abs/1901.00596>

---

**Thank you!**