THE UNIVERSITY OF BRITISH COLUMBIA

Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 18: Graph Neural Networks

Logistics

— All **slides** will be up **today**!

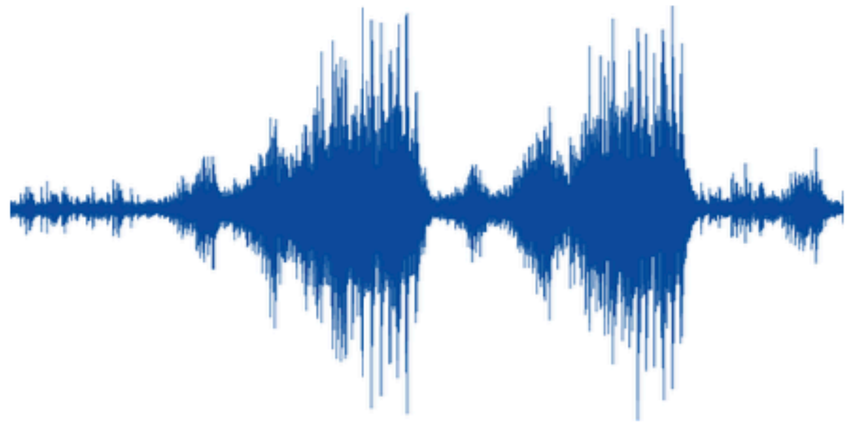
— Last **lecture** by me

— **Paper list** is up (volunteers)?

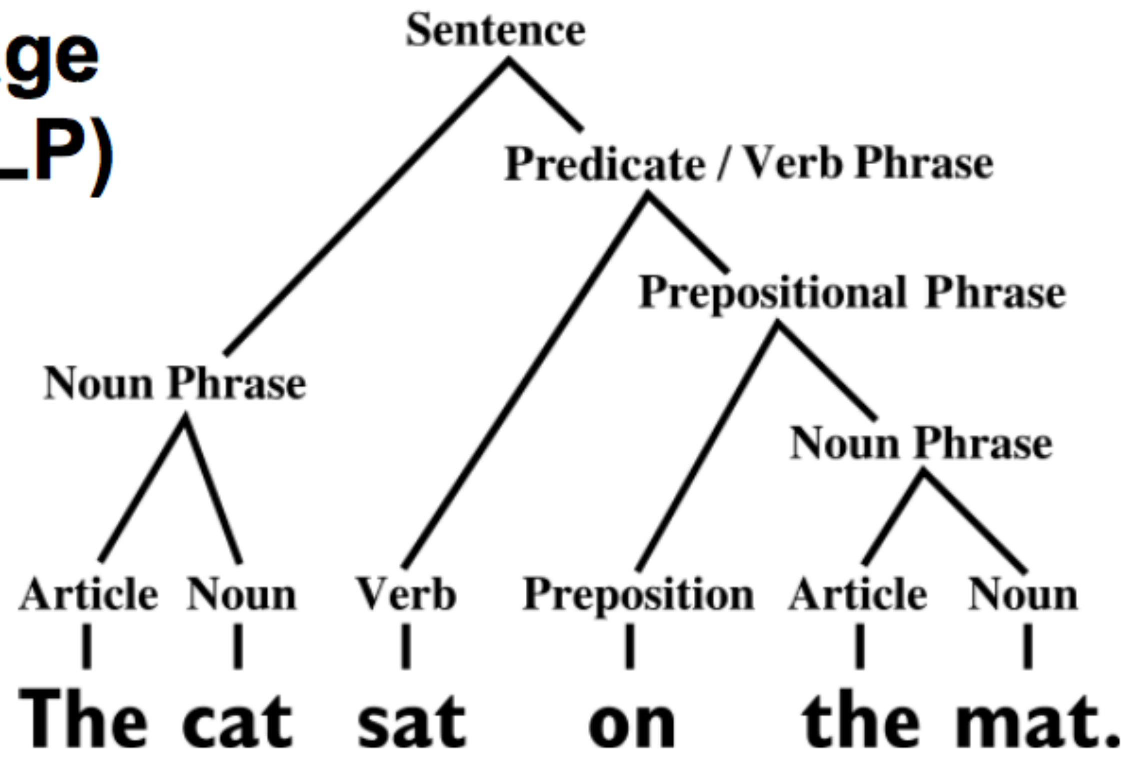
Traditional **Neural Networks**

IMAGENET

Speech data

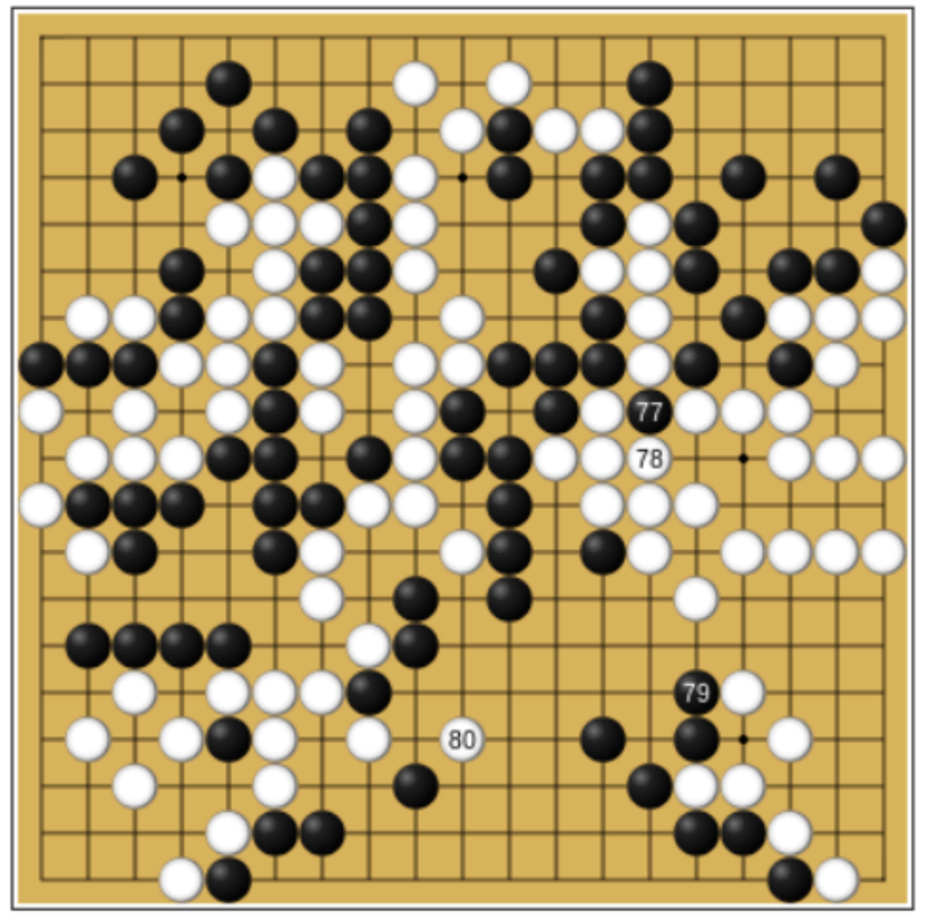


Natural language processing (NLP)



...

Grid games



Deep neural nets that exploit:

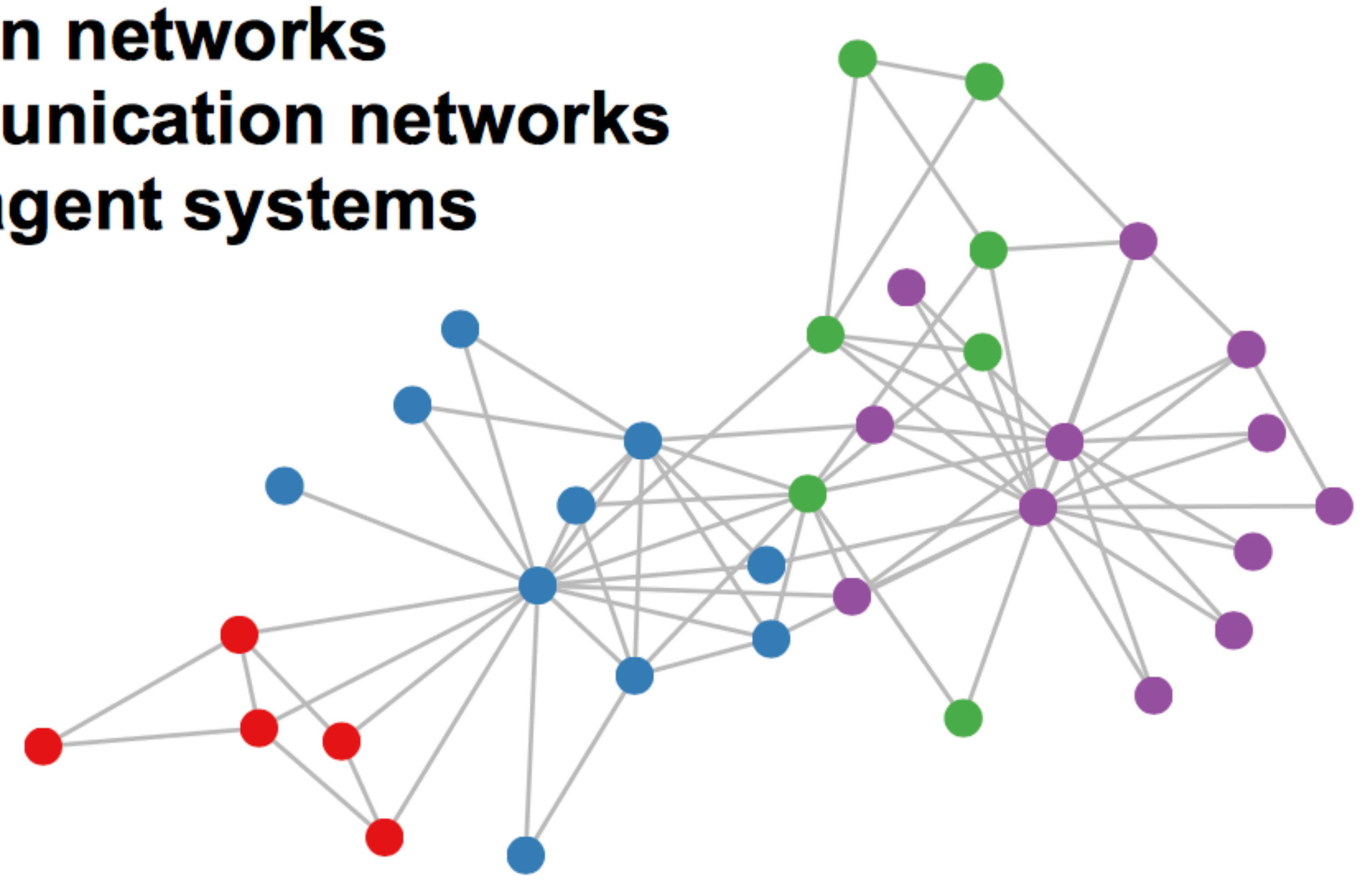
- translation equivariance (weight sharing)
- hierarchical compositionality

* slide from Thomas Kipf, **University of Amsterdam**

Graph-structured Data

A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

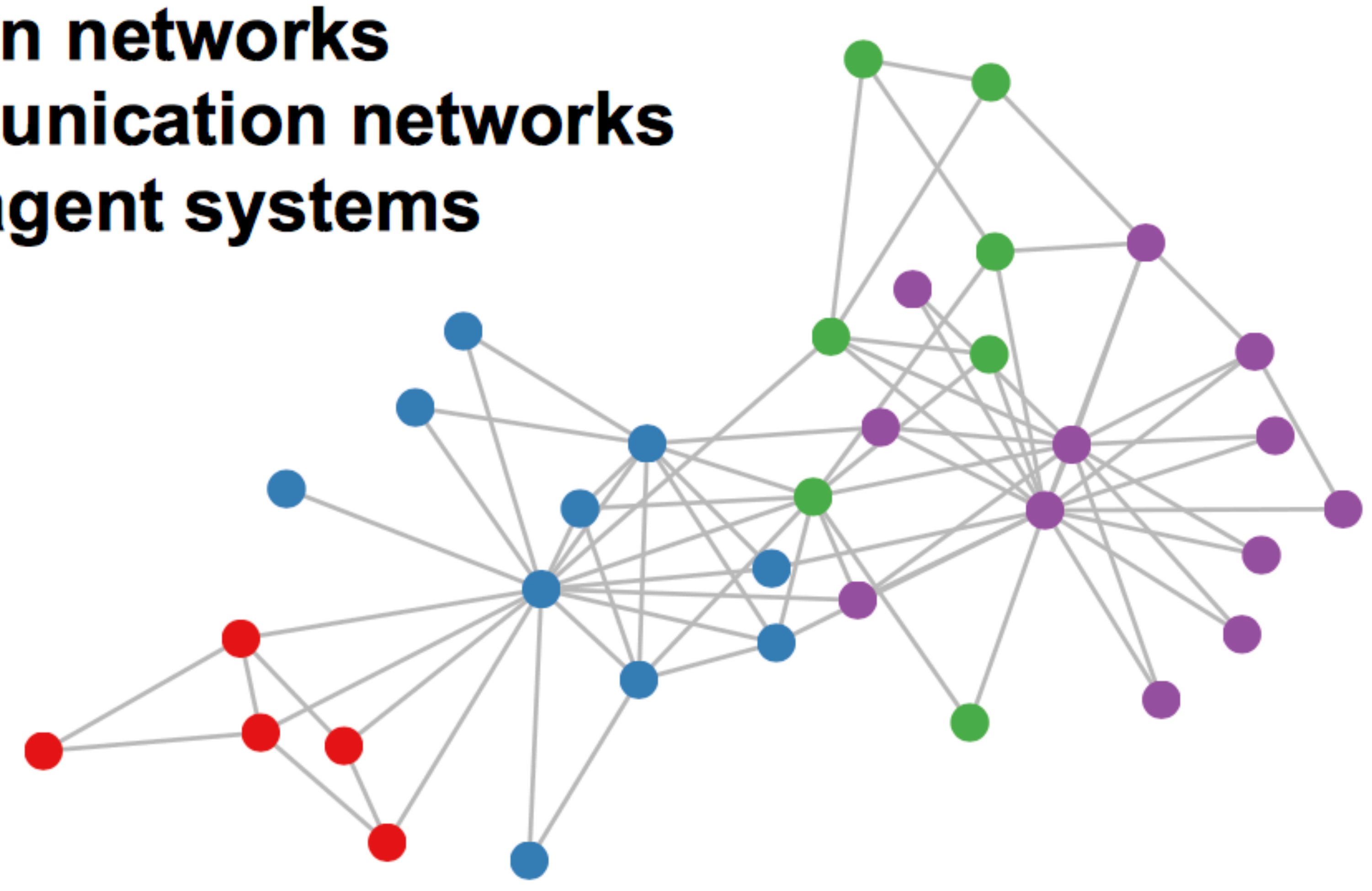


* slide from Thomas Kipf, **University of Amsterdam**

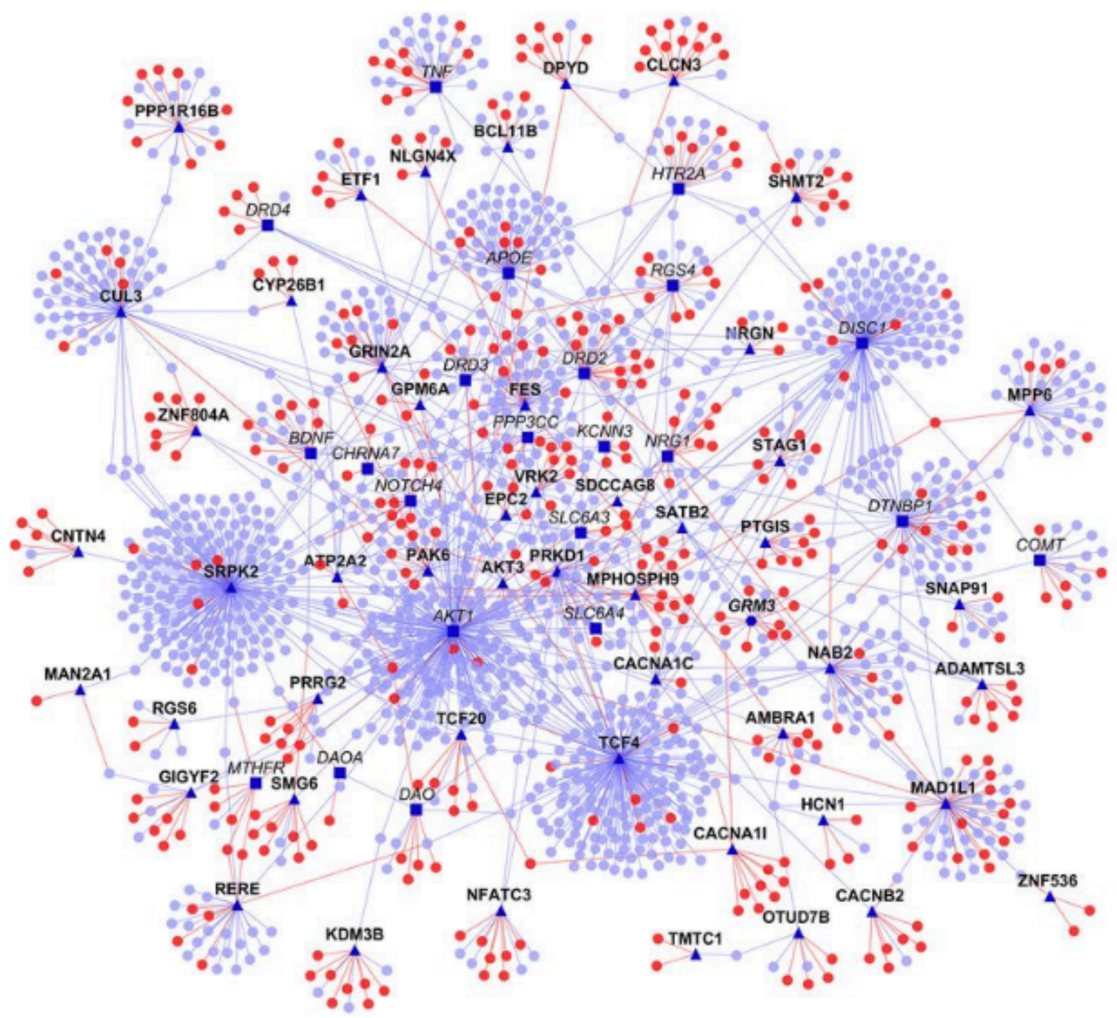
Graph-structured Data

A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems



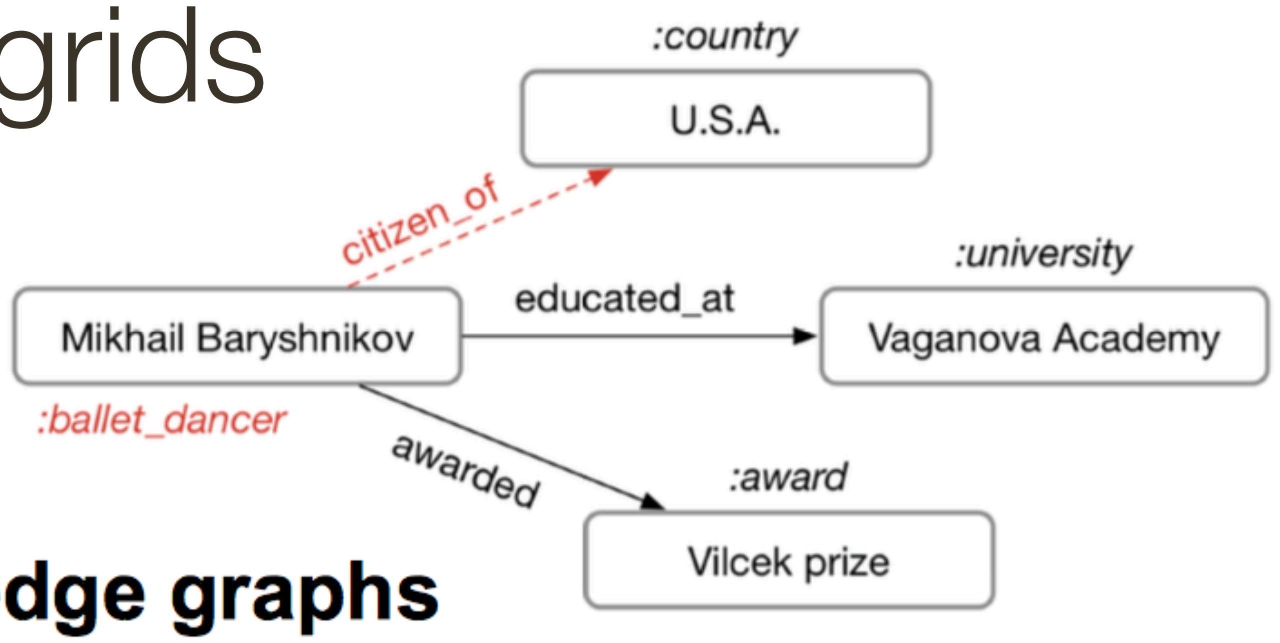
Protein interaction networks



* slide from Thomas Kipf, **University of Amsterdam**

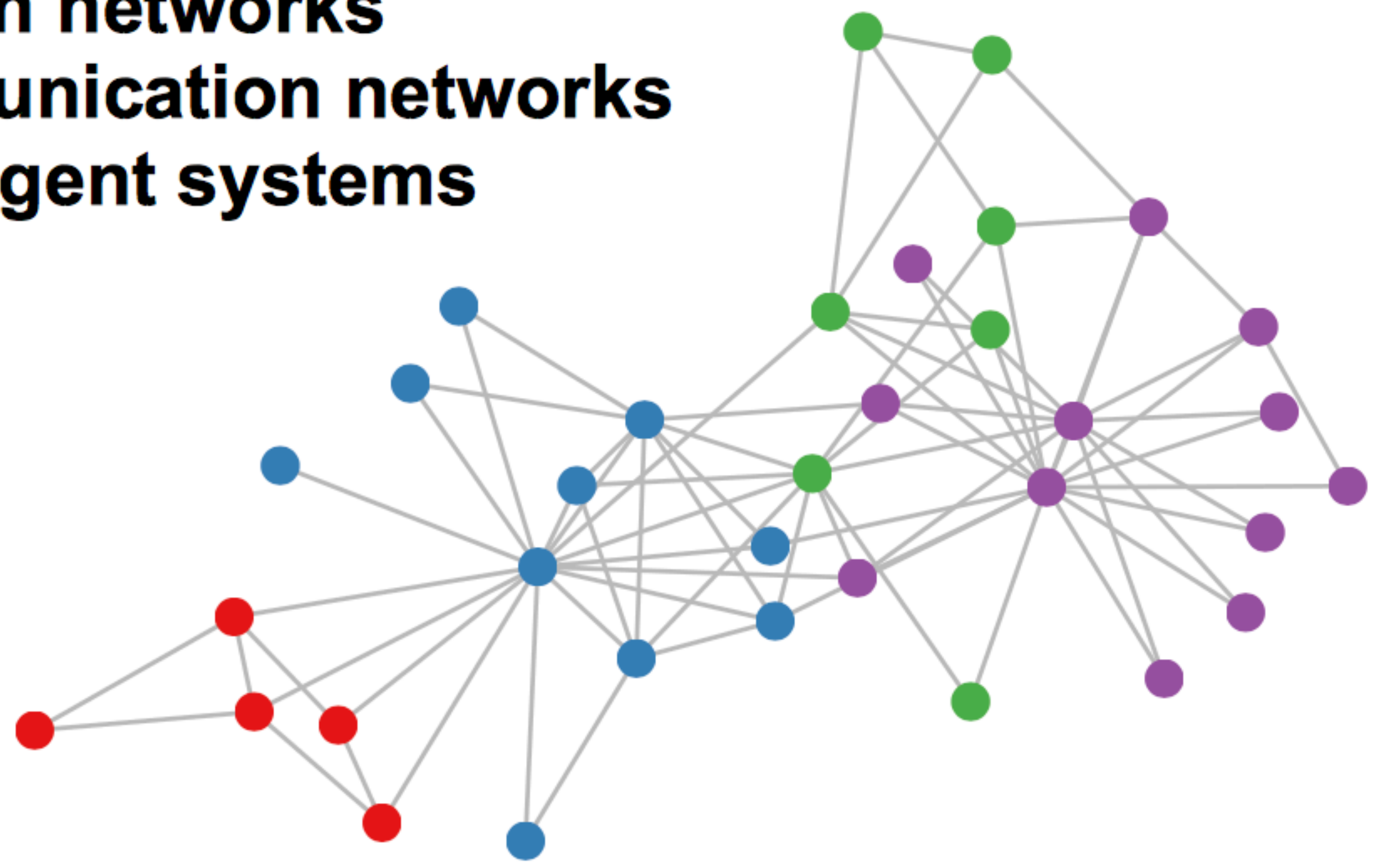
Graph-structured Data

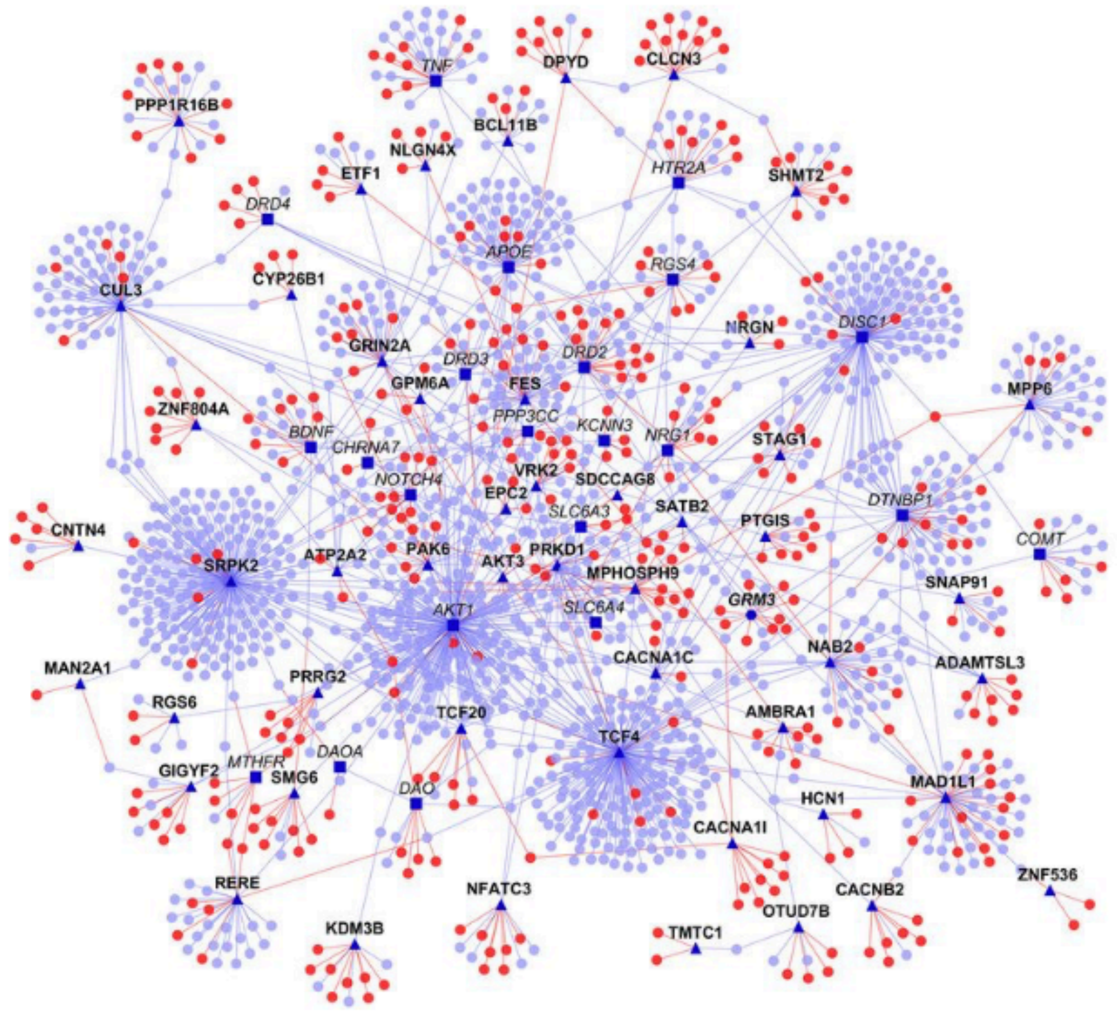
A lot of real-world data does not “live” on grids



Knowledge graphs

Social networks
Citation networks
Communication networks
Multi-agent systems





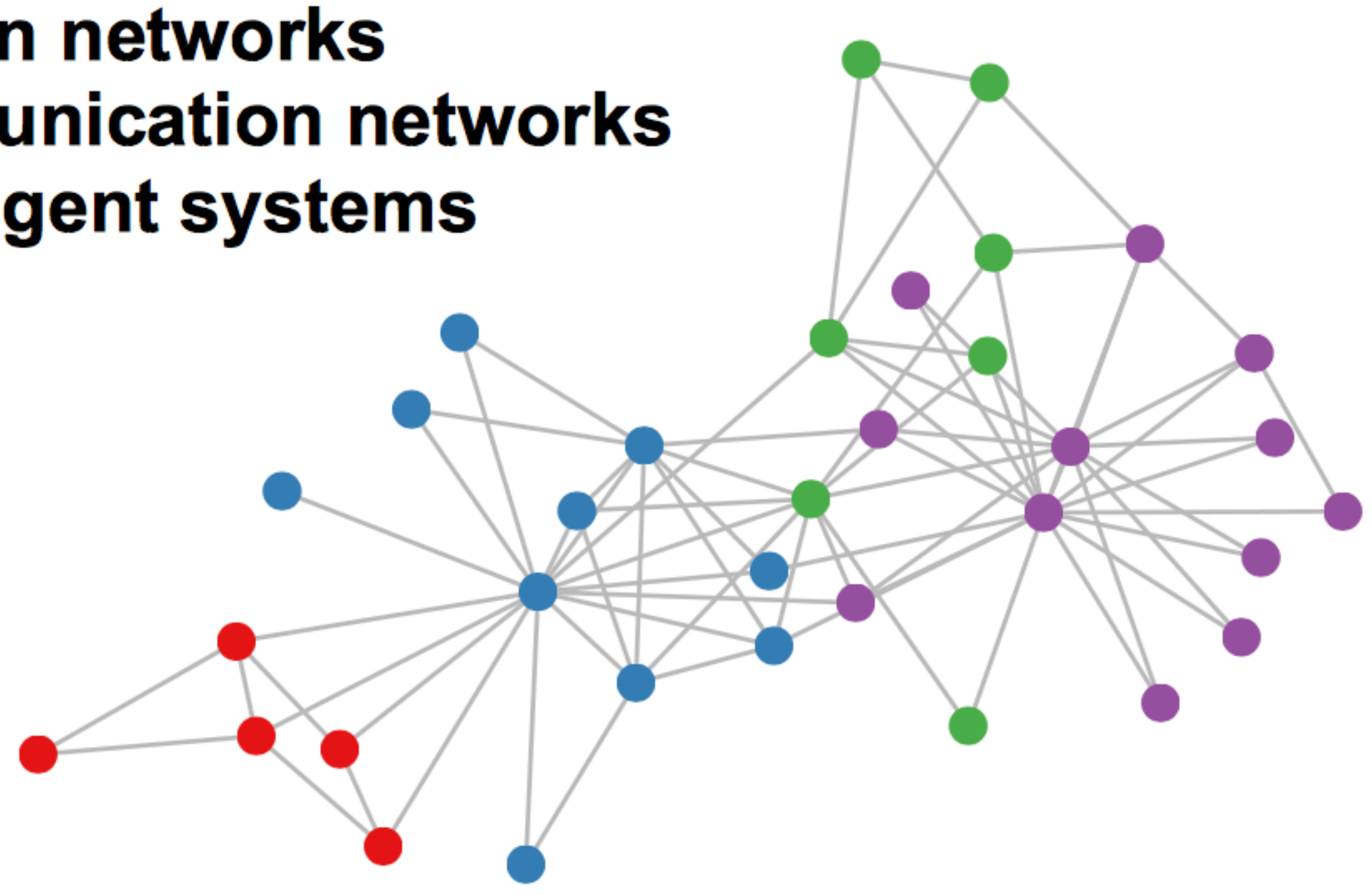
Protein interaction networks

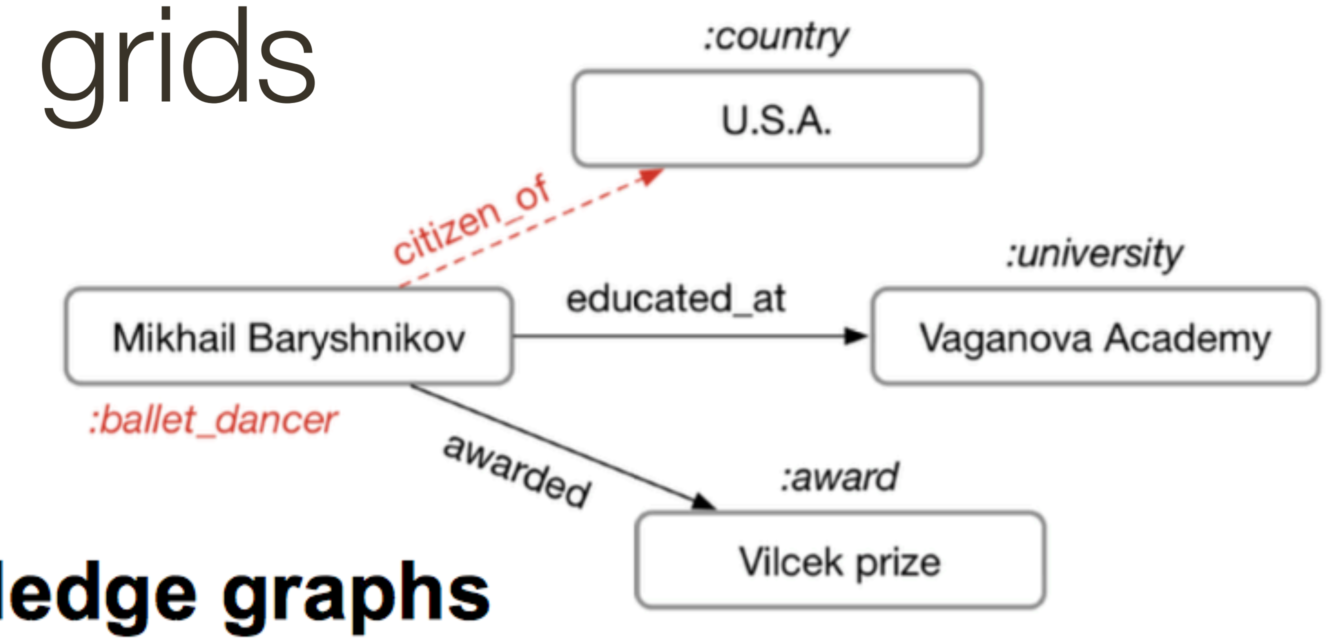
* slide from Thomas Kipf, **University of Amsterdam**

Graph-structured Data

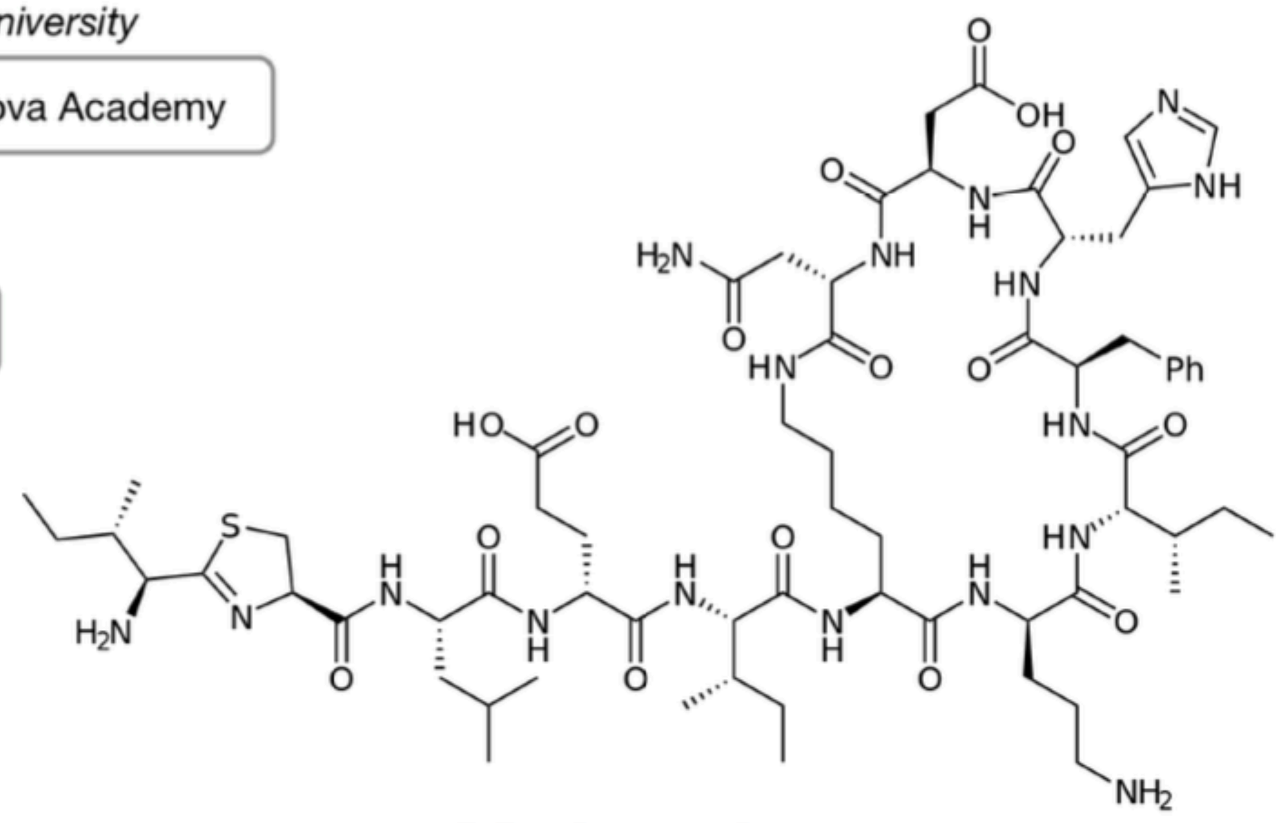
A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

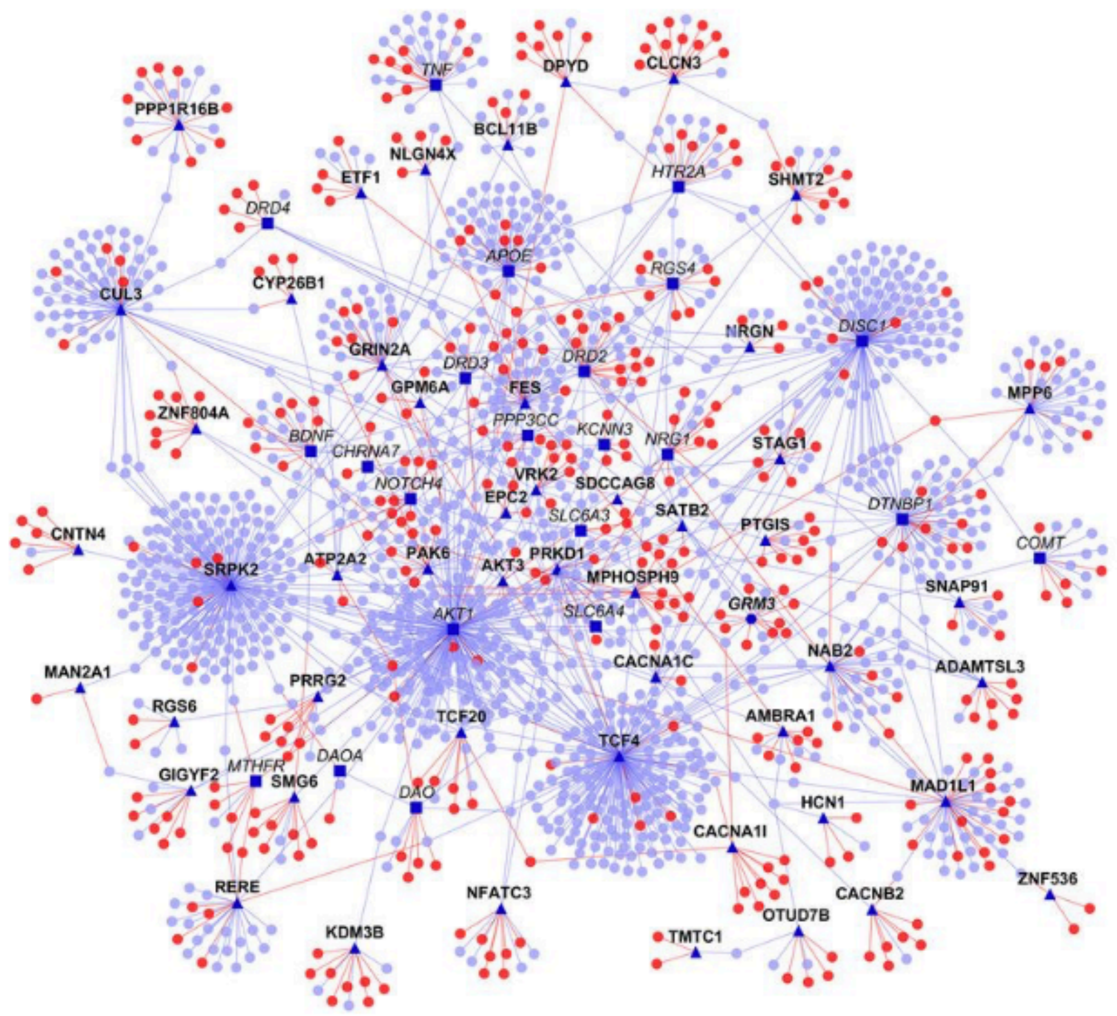




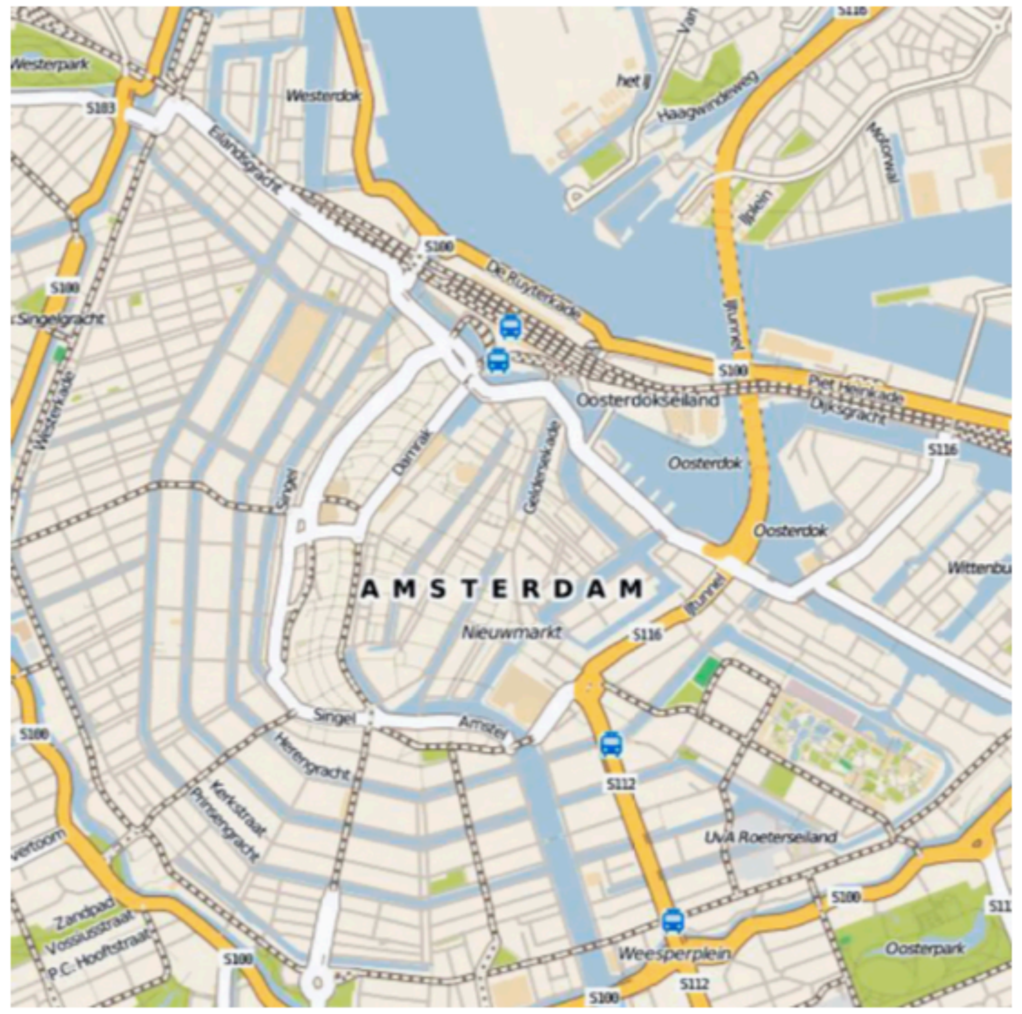
Knowledge graphs



Molecules



Protein interaction networks



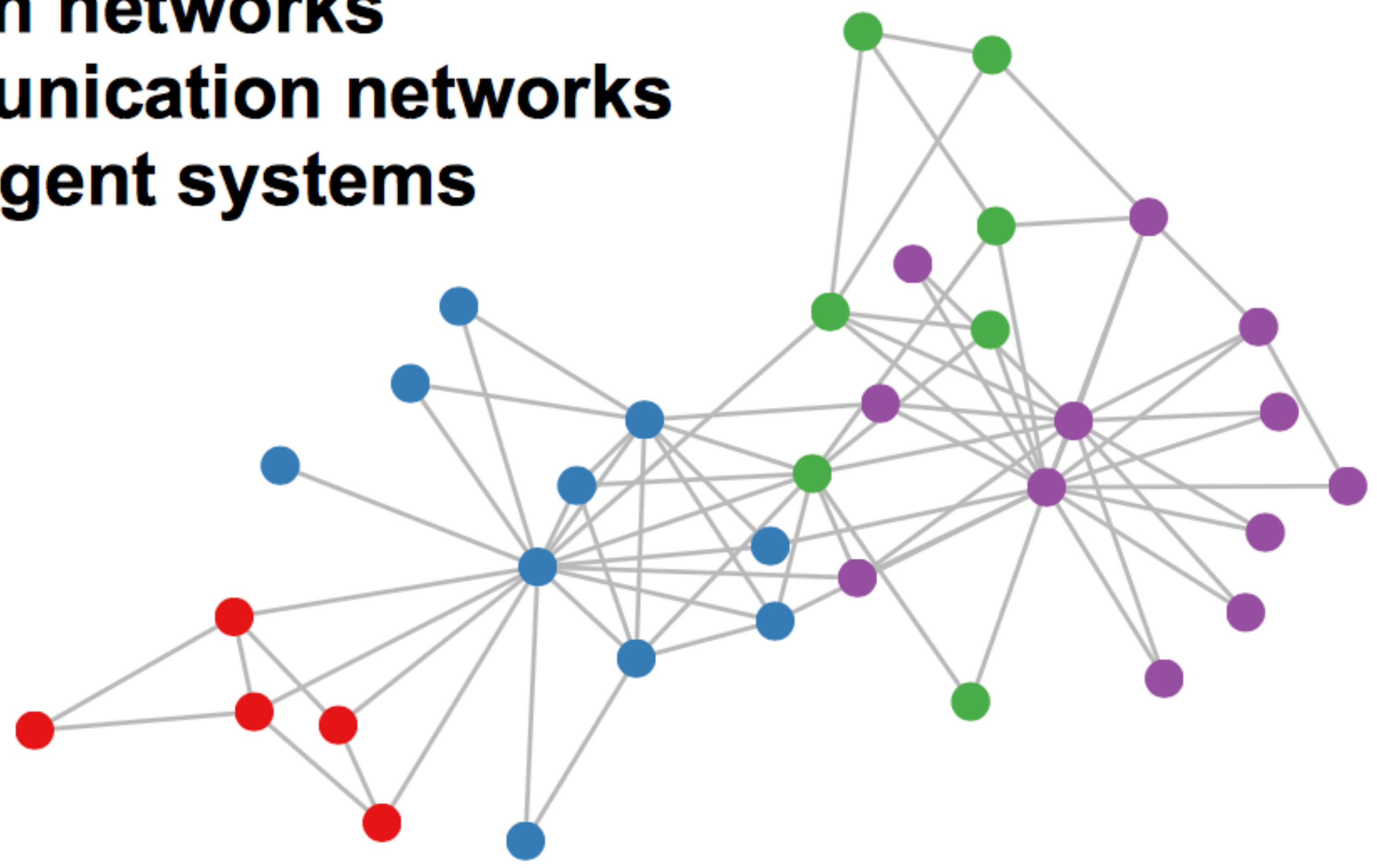
Road maps

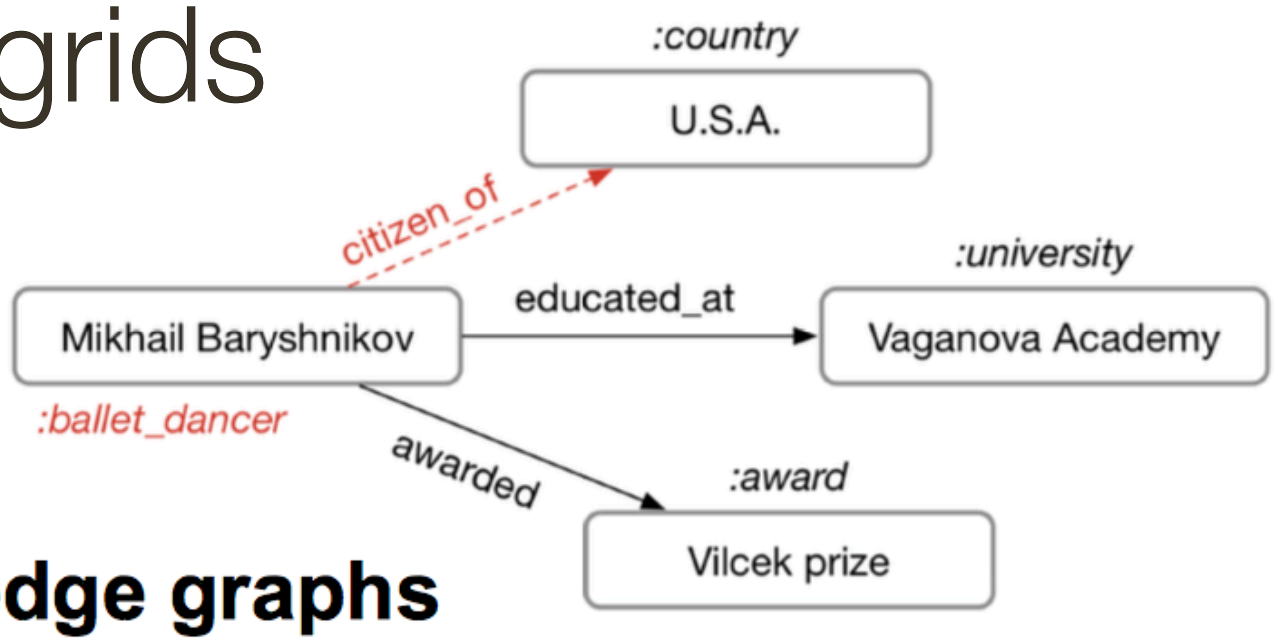
* slide from Thomas Kipf, **University of Amsterdam**

Graph-structured Data

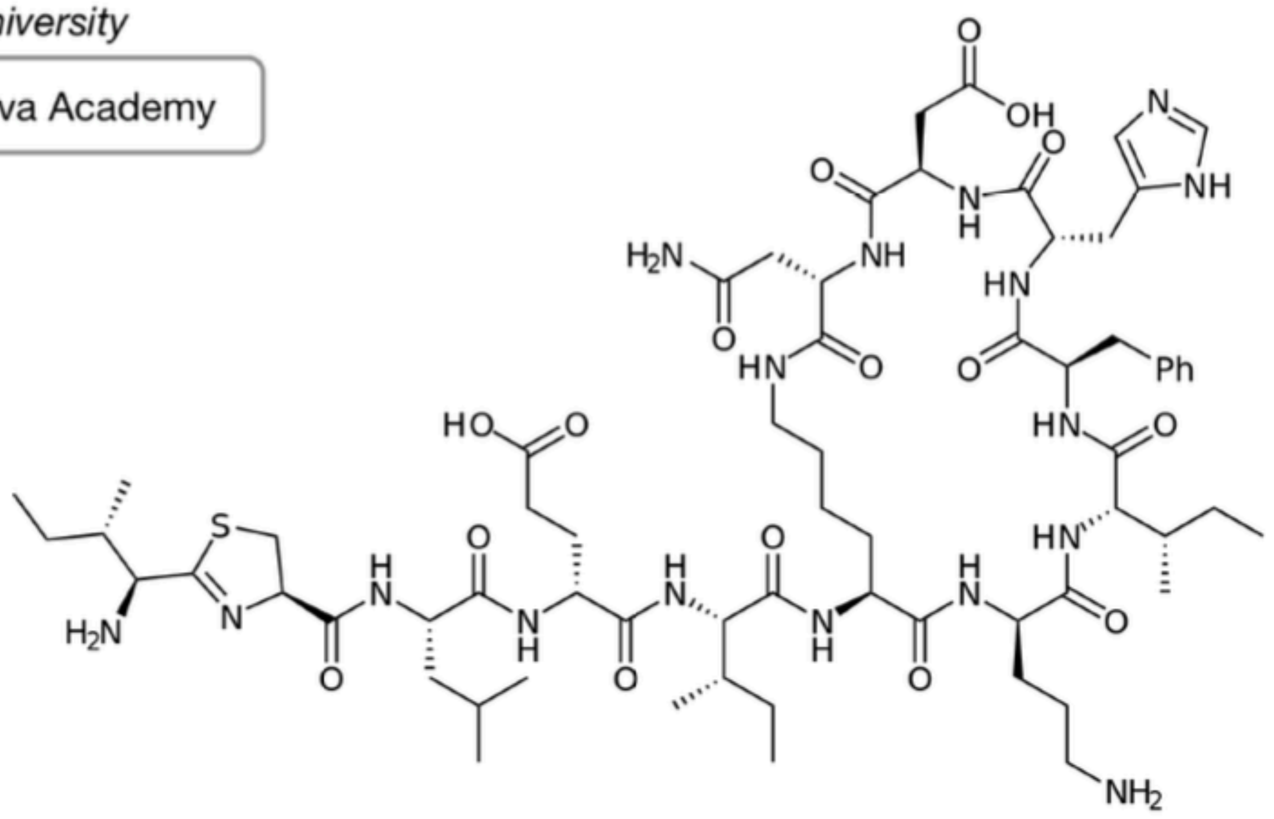
A lot of real-world data does not "live" on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

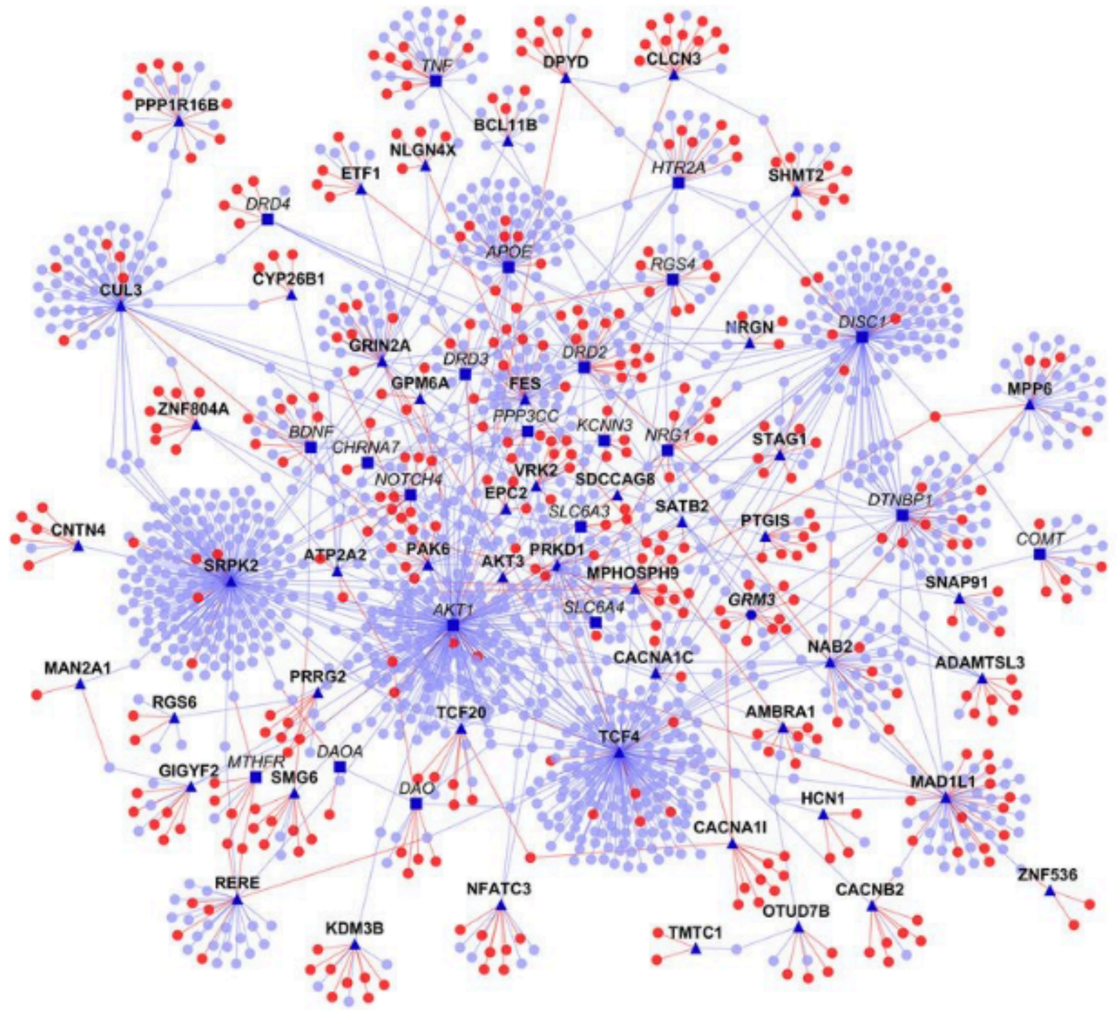




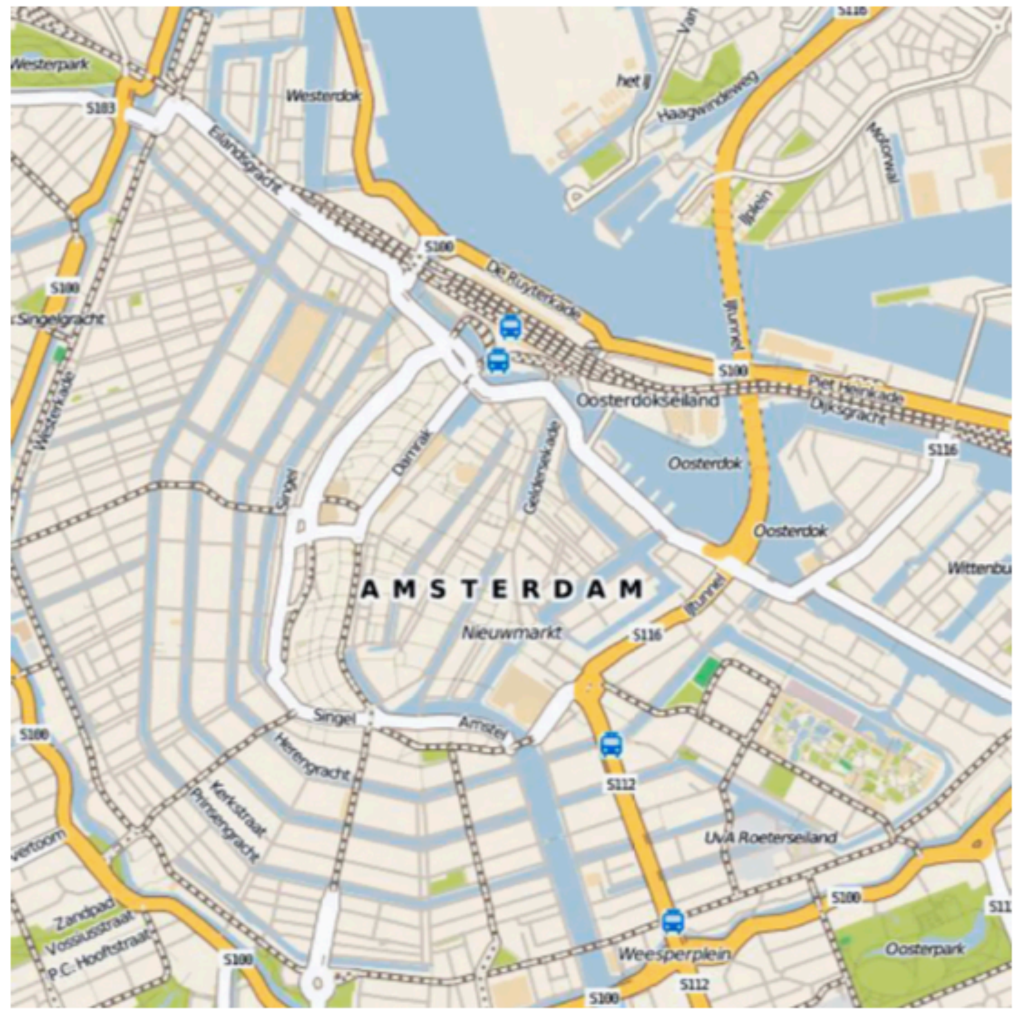
Knowledge graphs



Molecules



Protein interaction networks

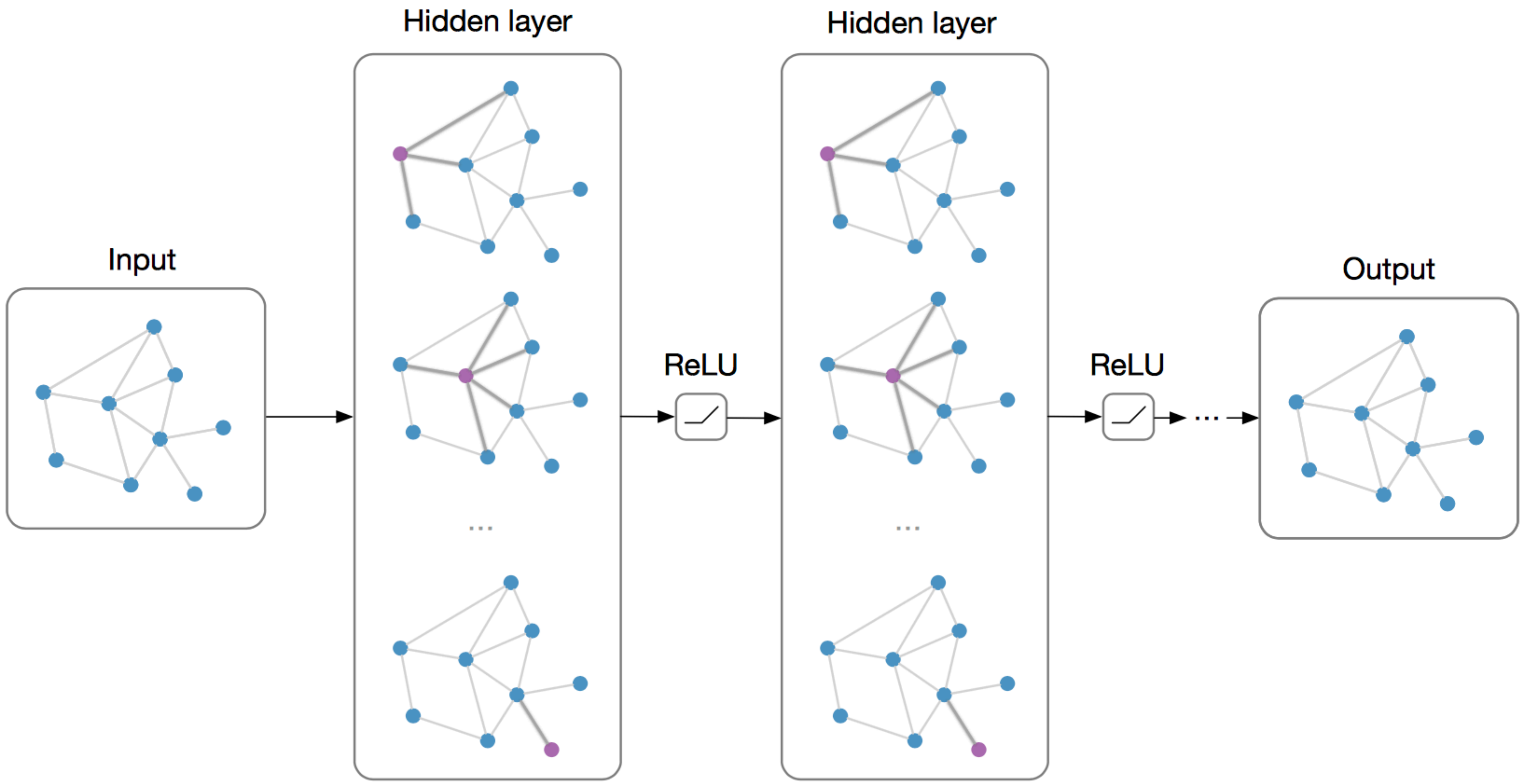


Road maps

Standard **CNN** and **RNN** architectures don't work on this data

* slide from Thomas Kipf, **University of Amsterdam**

Graph Neural Networks (GNNs)



Main Idea: Pass massages between pairs of nodes and agglomerate

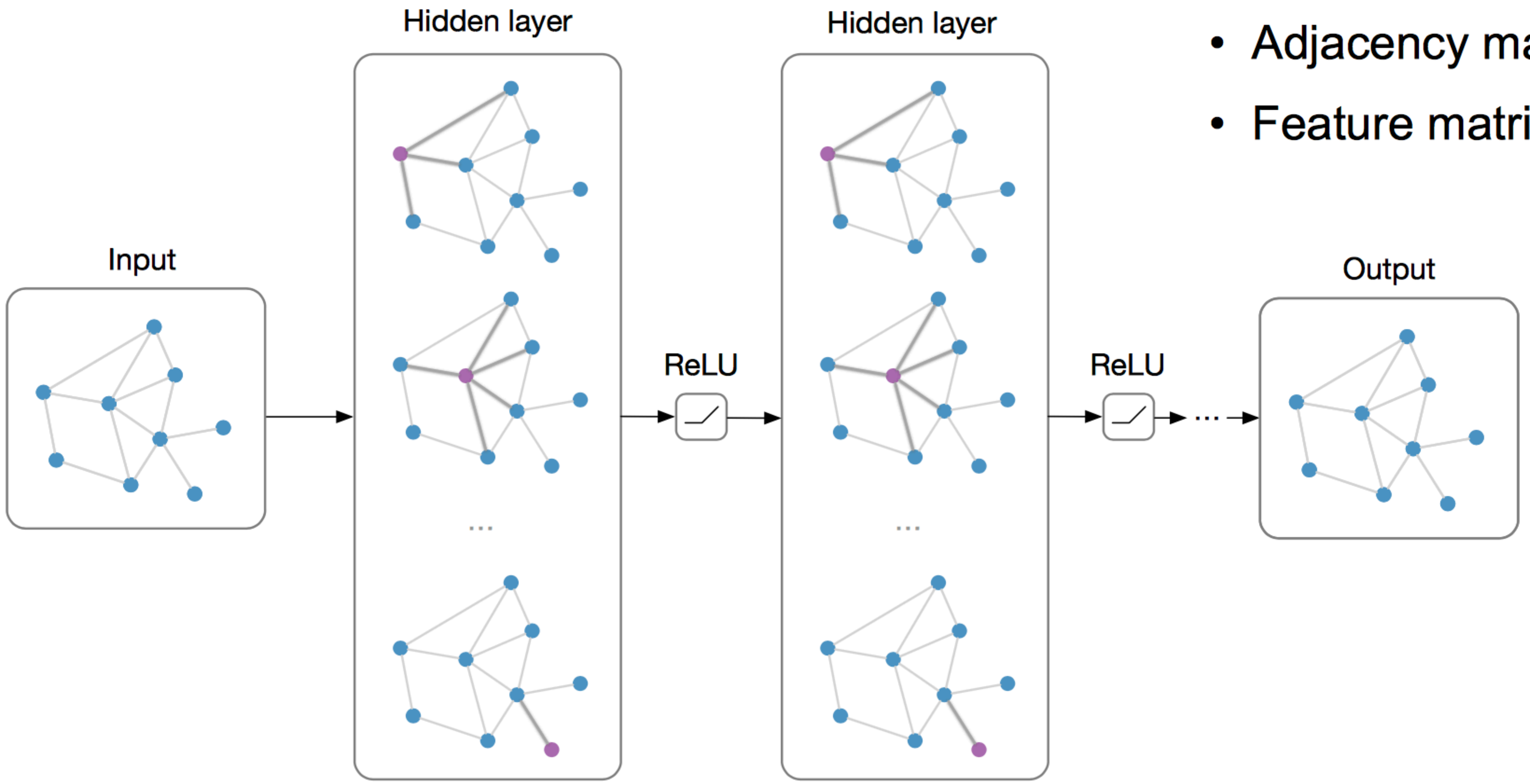
Alternative Interpretation: Pass massages between nodes to refine node (and possibly edge) representations

* slide from Thomas Kipf, **University of Amsterdam**

Graph Neural Networks (GNNs)

Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$



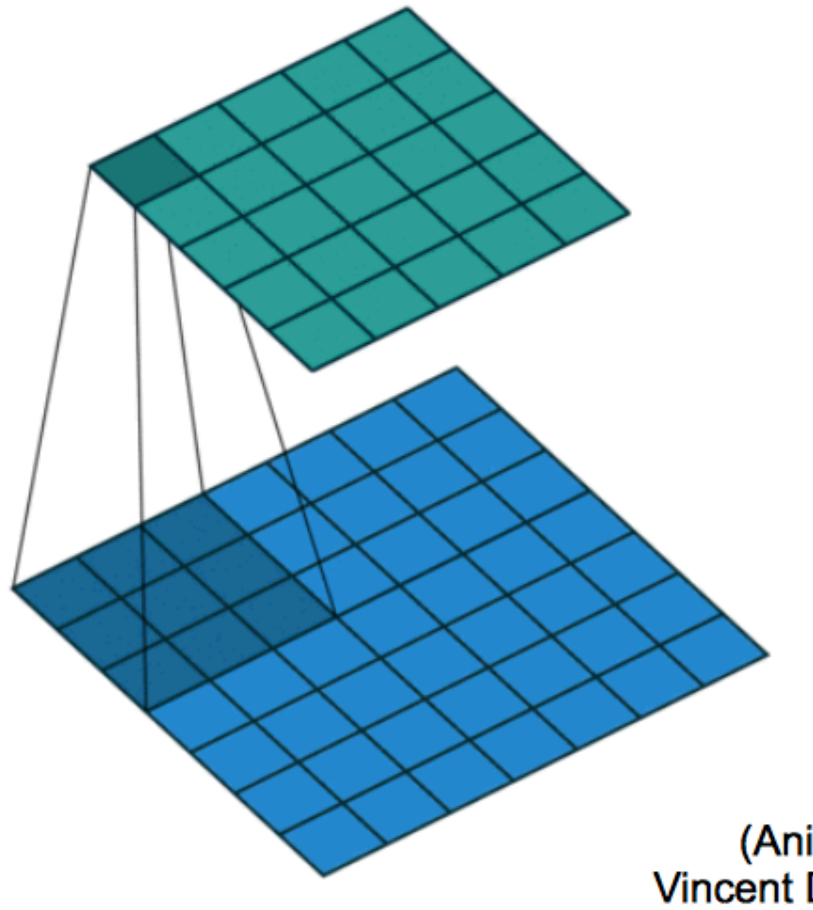
Main Idea: Pass massages between pairs of nodes and agglomerate

Alternative Interpretation: Pass massages between nodes to refine node (and possibly edge) representations

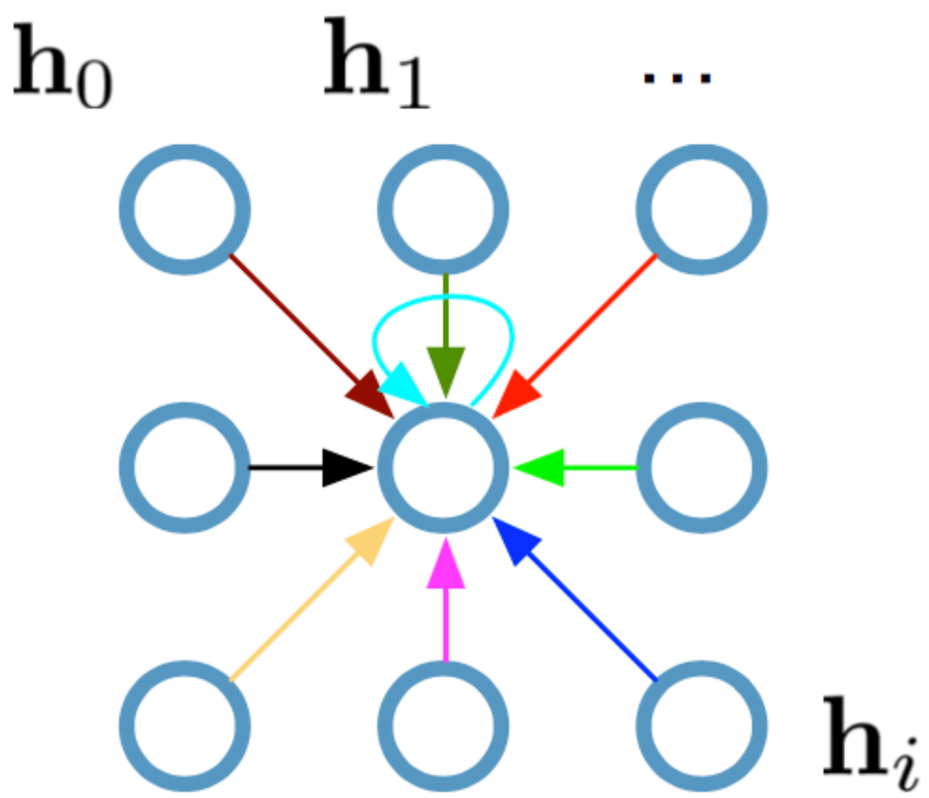
* slide from Thomas Kipf, **University of Amsterdam**

Recap: Convolutional Neural Networks (CNNs) on Grids

Single CNN layer with 3x3 filter:



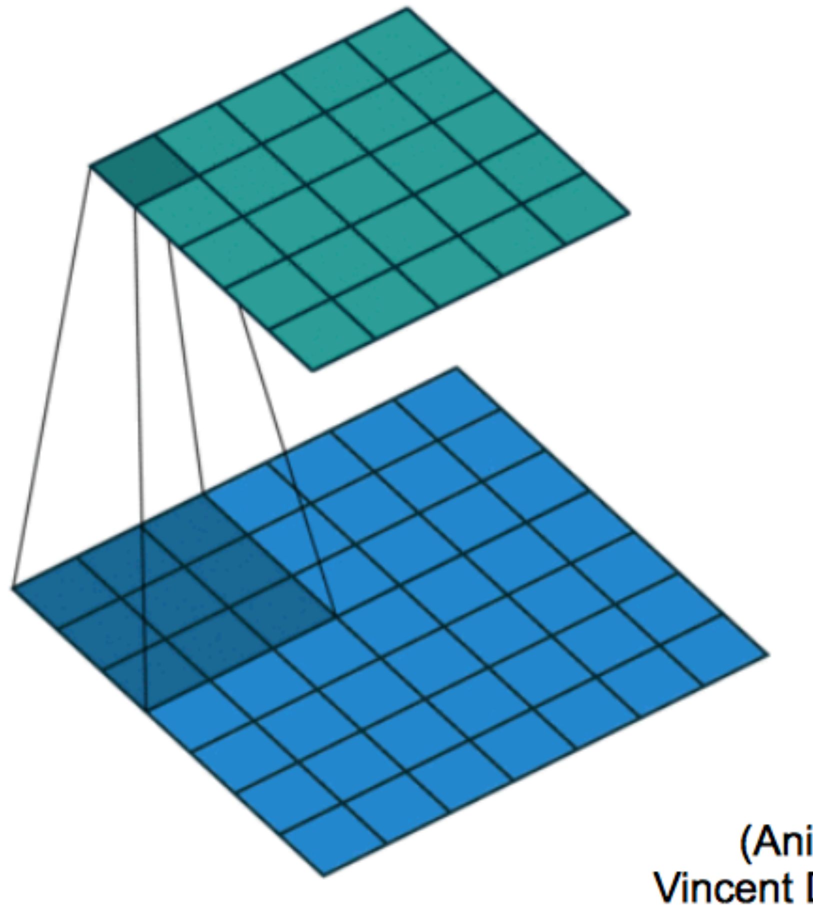
(Animation by Vincent Dumoulin)



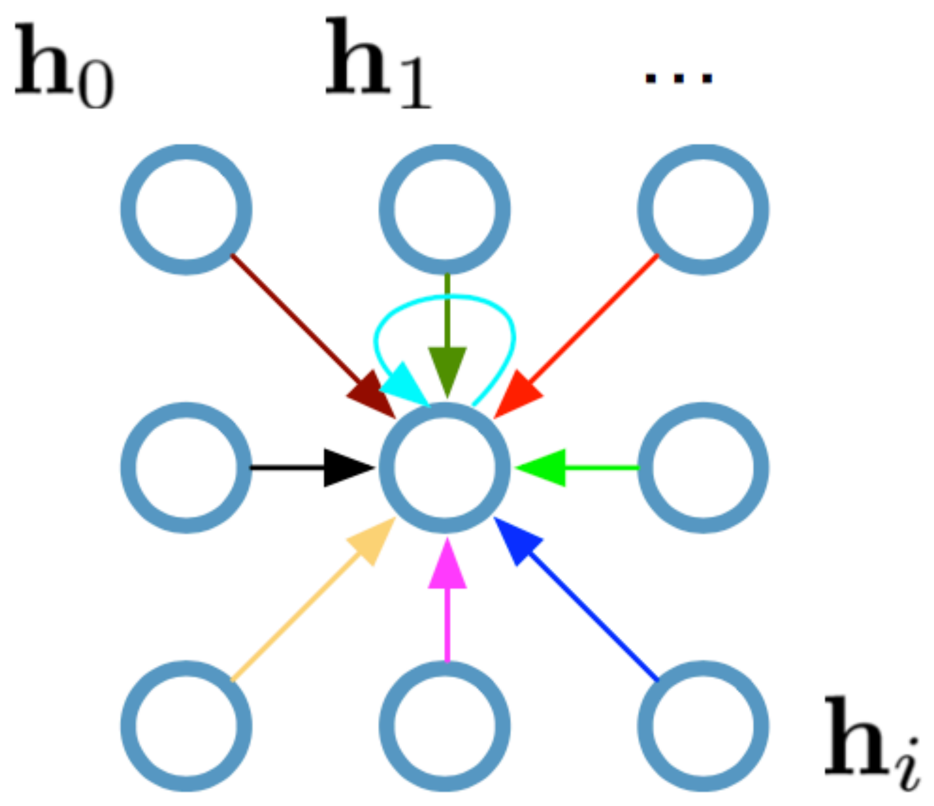
* slide from Thomas Kipf, **University of Amsterdam**

Recap: Convolutional Neural Networks (CNNs) on Grids

Single CNN layer with 3x3 filter:



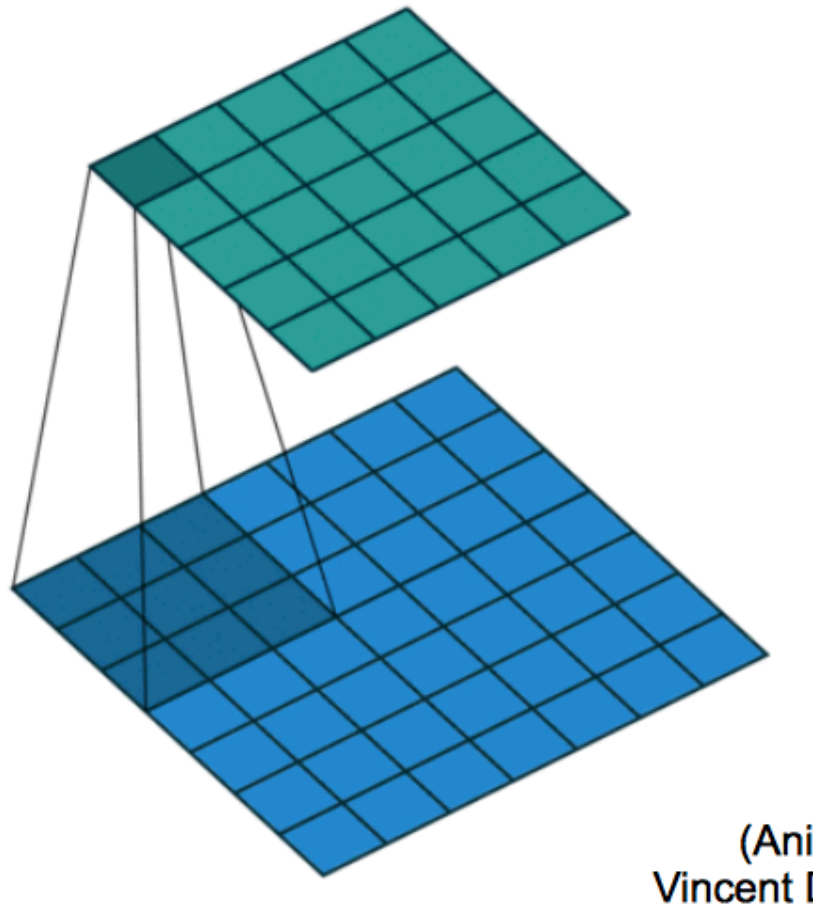
(Animation by Vincent Dumoulin)



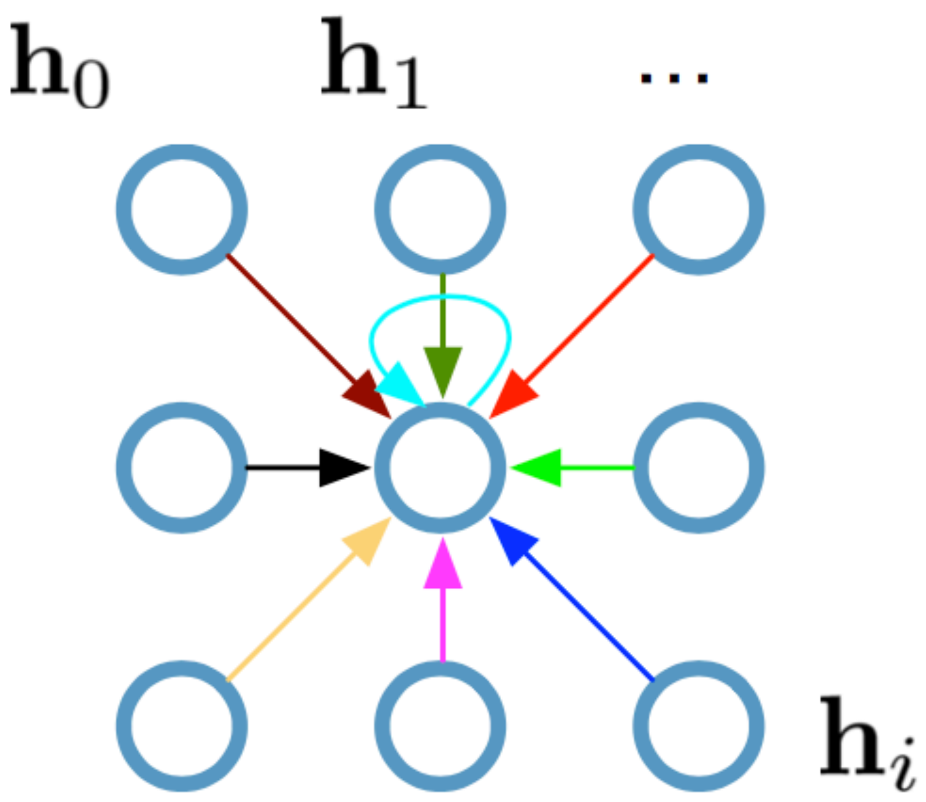
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Recap: Convolutional Neural Networks (CNNs) on Grids

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



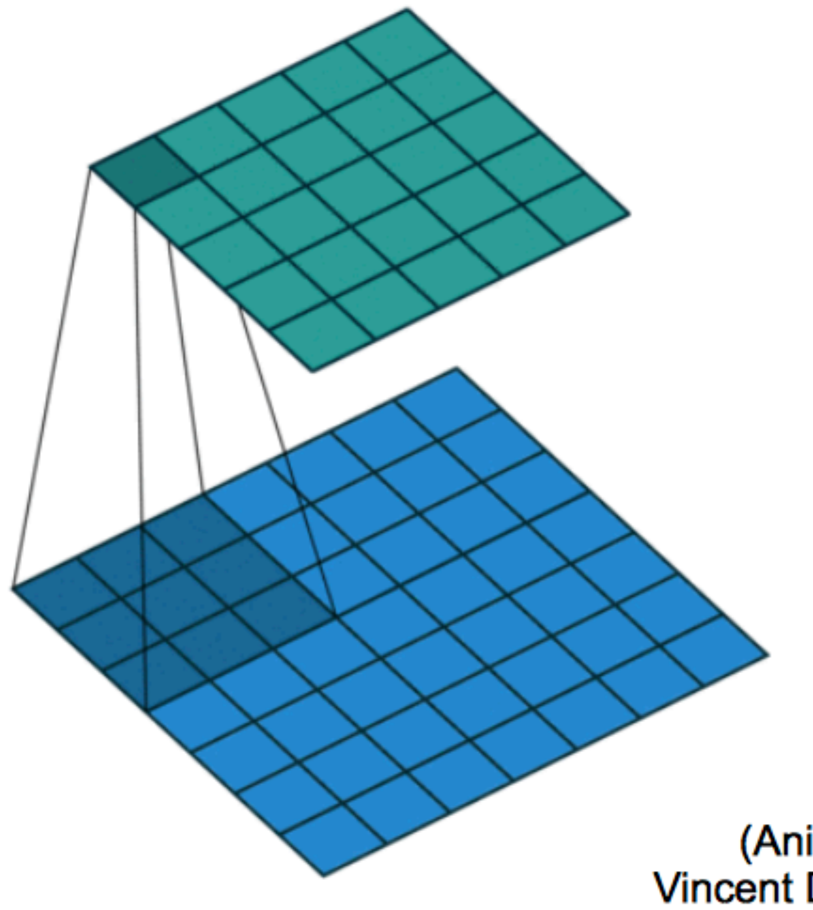
Update for a single pixel:

- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

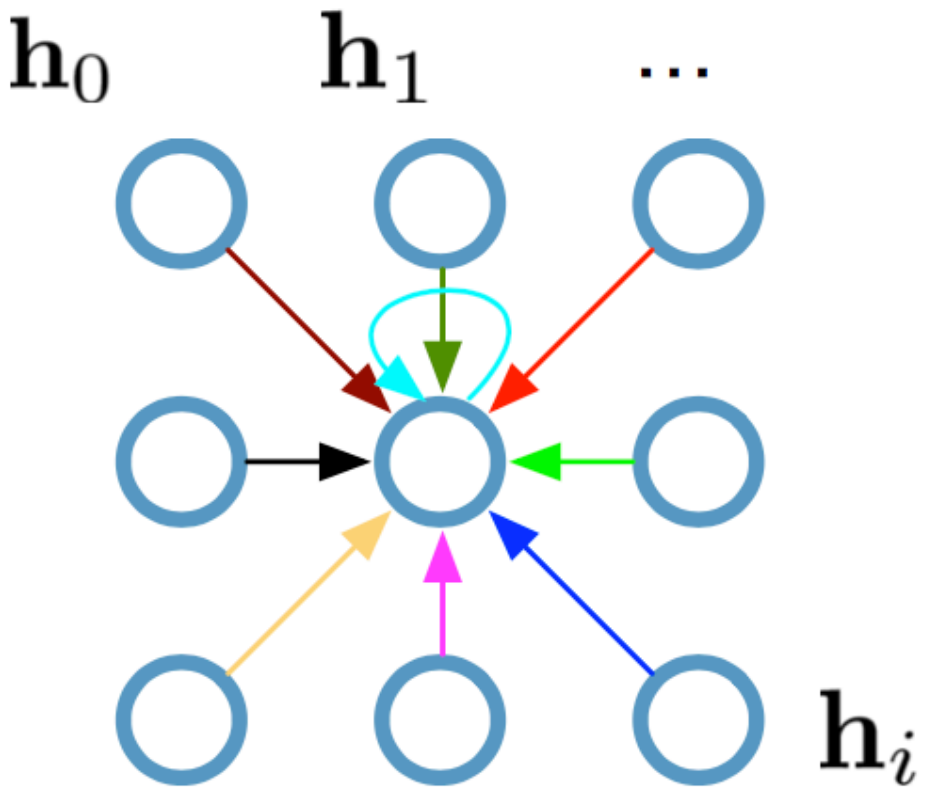
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Recap: Convolutional Neural Networks (CNNs) on Grids

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

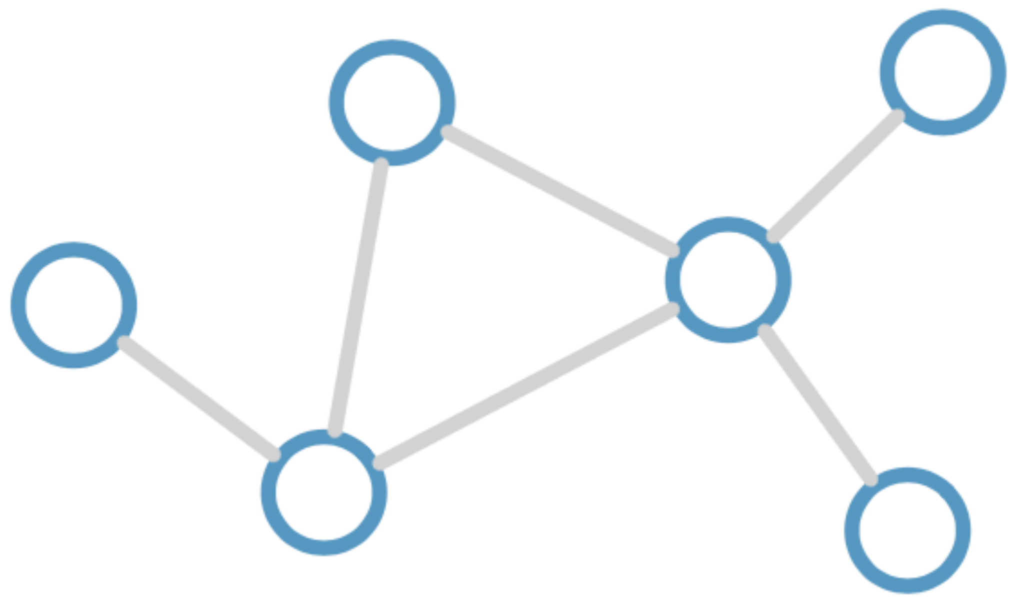
Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

Graph Convolutional Networks (GCNs)

Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Consider this
undirected graph:

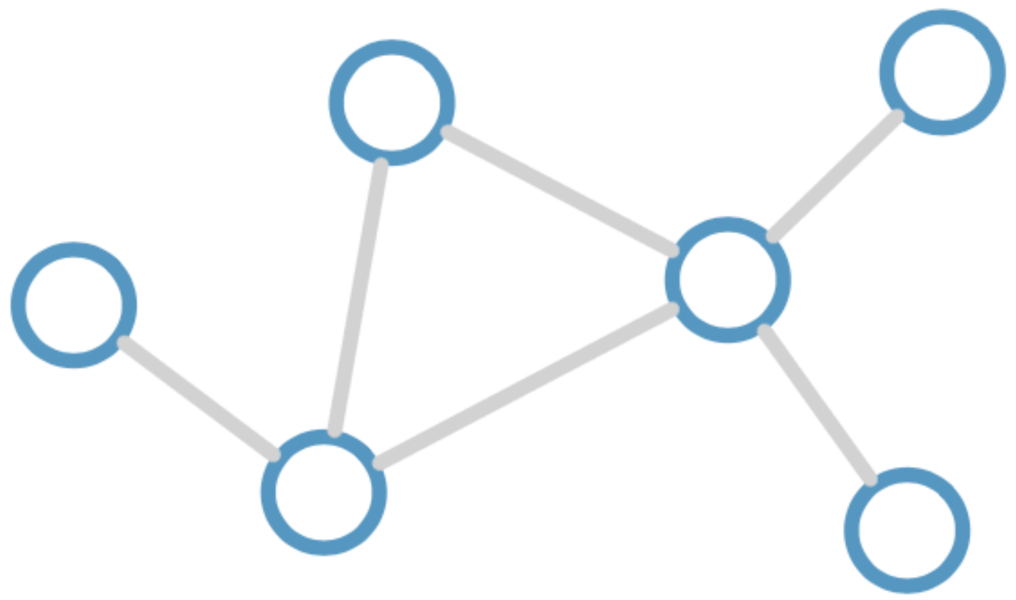


* slide from Thomas Kipf, **University of Amsterdam**

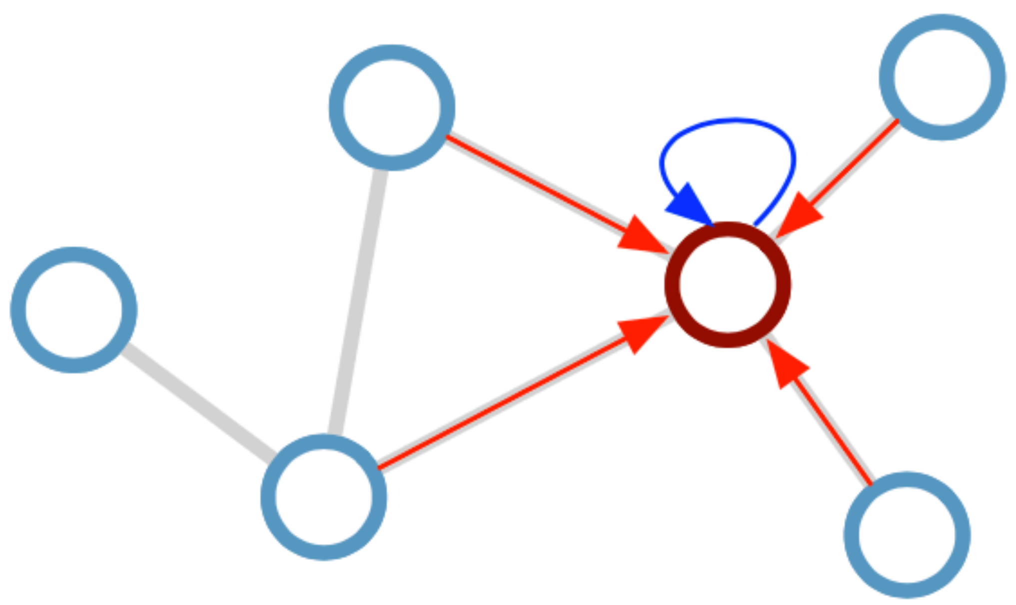
Graph Convolutional Networks (GCNs)

Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Consider this undirected graph:



Calculate update for node in red:

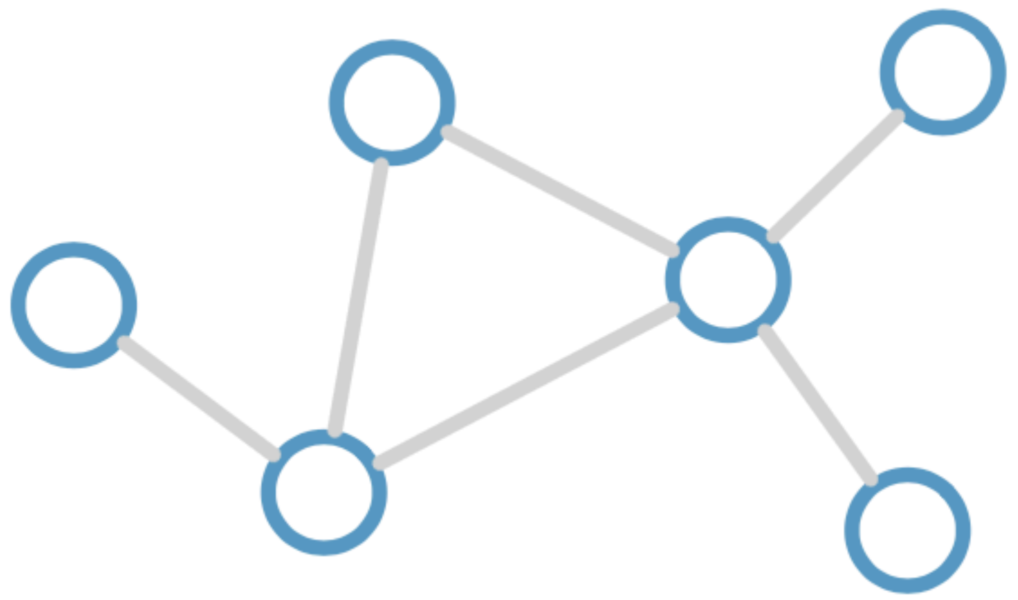


* slide from Thomas Kipf, **University of Amsterdam**

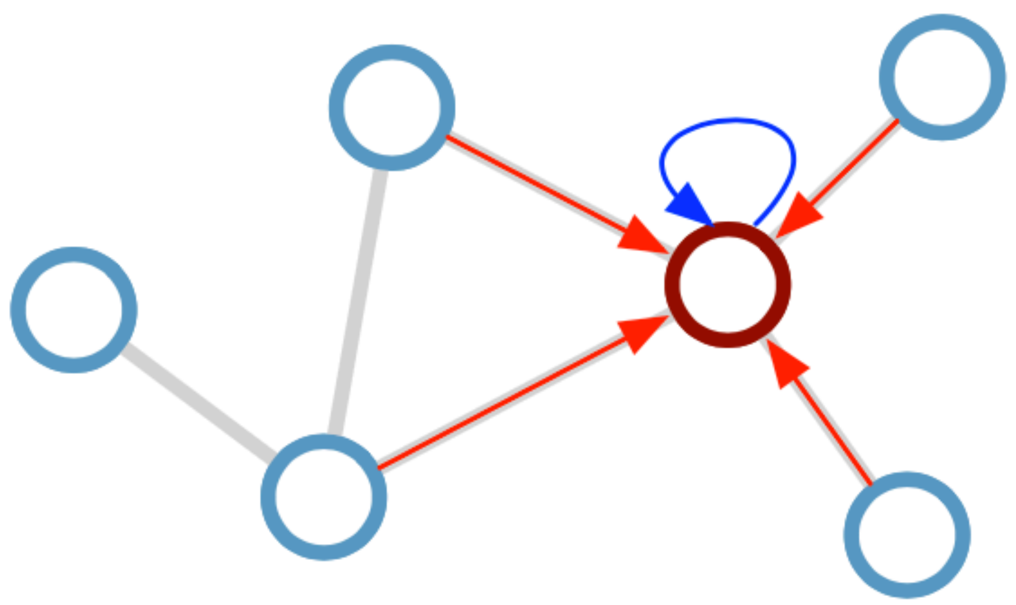
Graph Convolutional Networks (GCNs)

Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

$\mathcal{N}_i$: neighbor indices

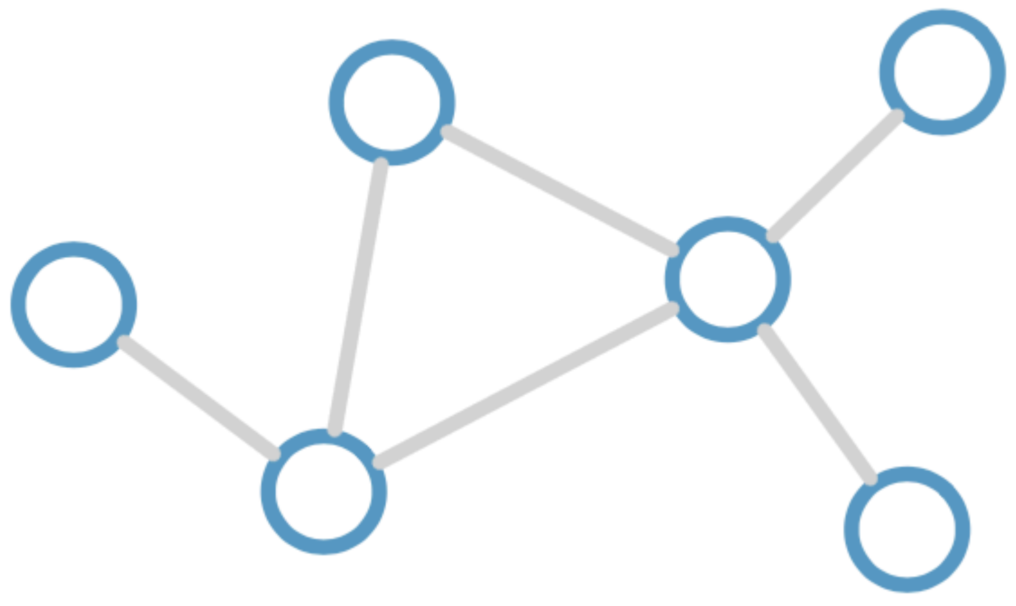
c_{ij} : norm. constant (fixed/trainable)

* slide from Thomas Kipf, **University of Amsterdam**

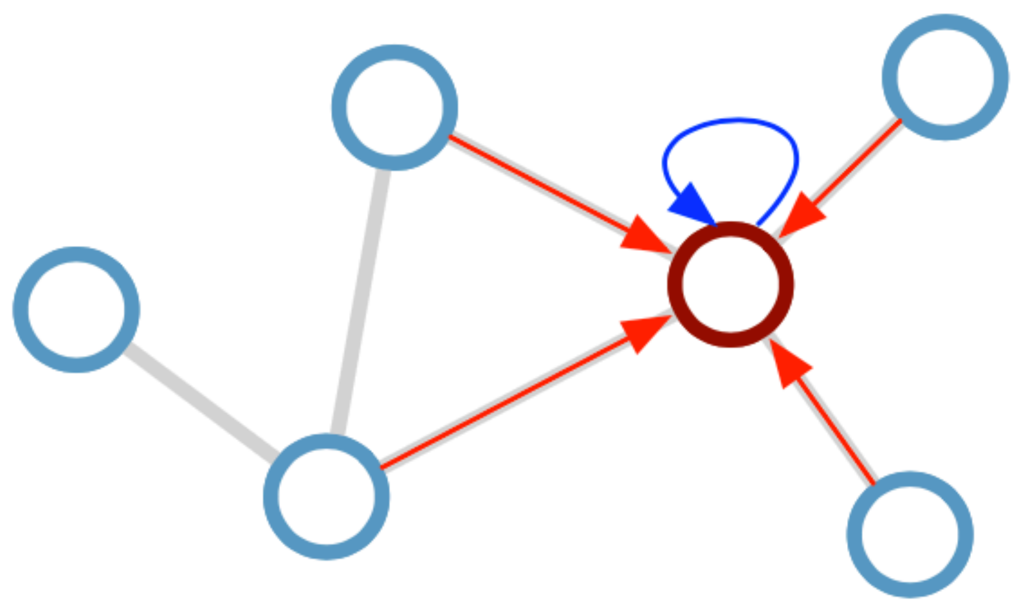
Graph Convolutional Networks (GCNs)

Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Consider this undirected graph:



Calculate update for node in red:



Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transductive and inductive settings

Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

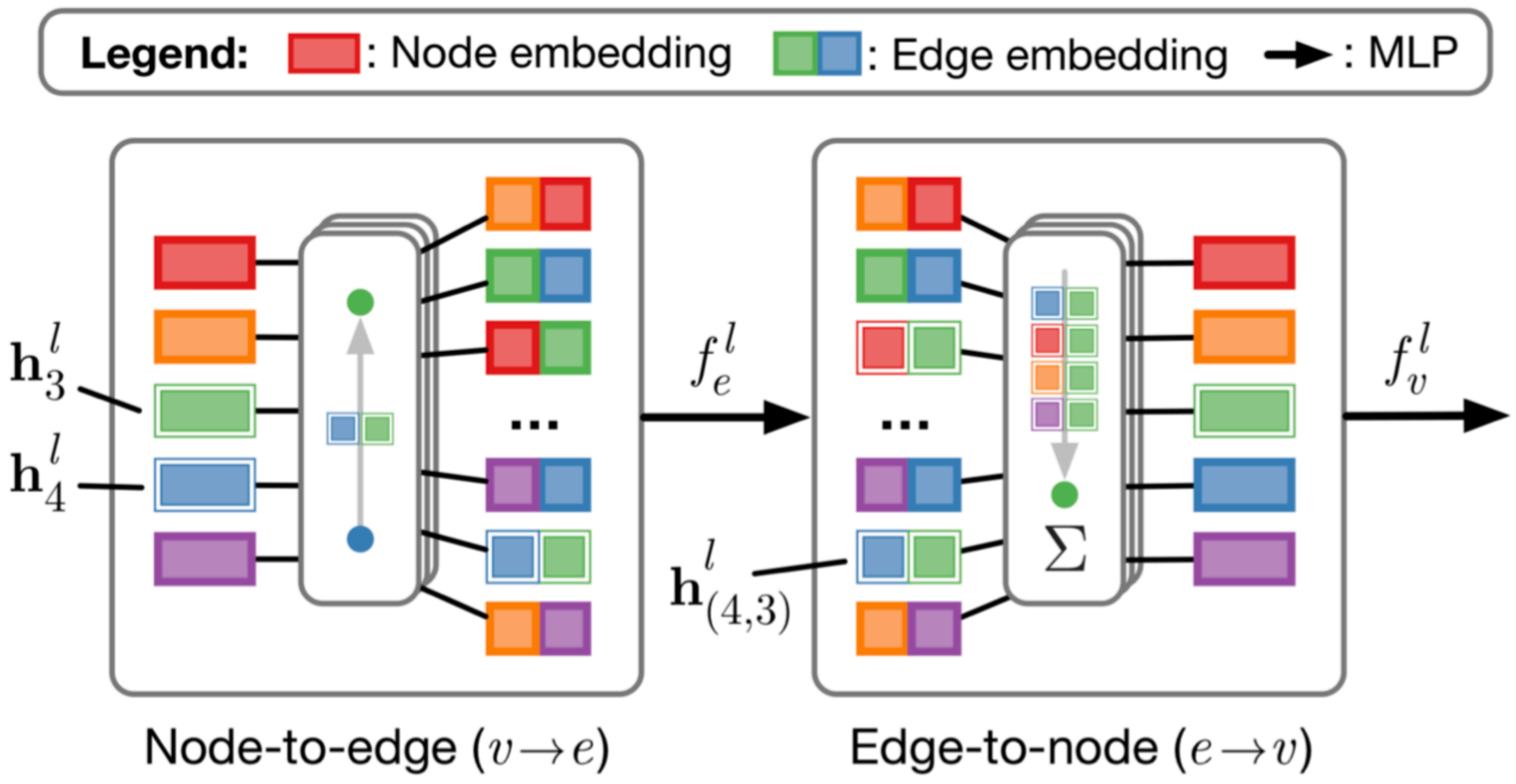
$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

* slide from Thomas Kipf, **University of Amsterdam**

GNNs with **Edge** Embeddings

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)



Formally:

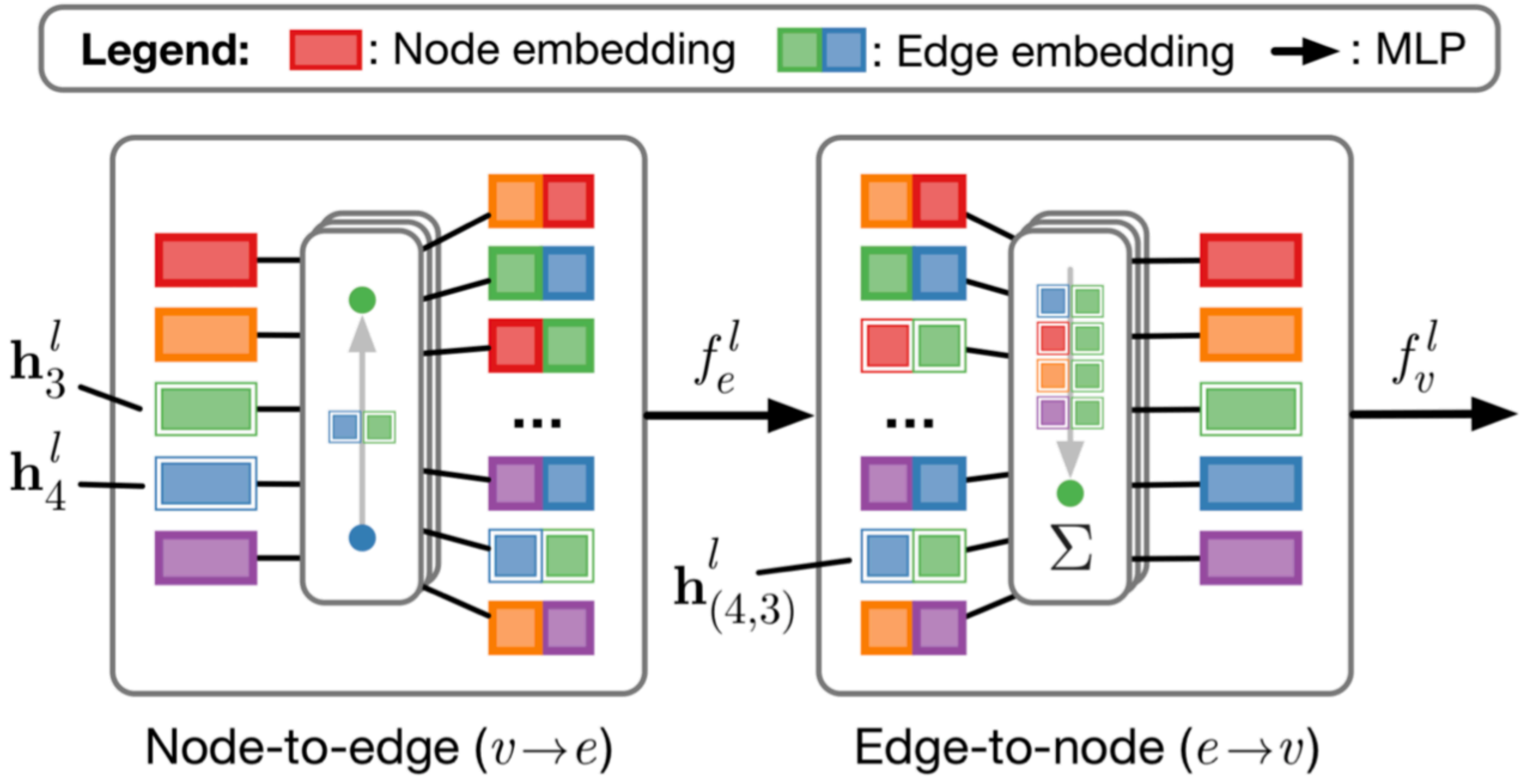
$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

* slide from Thomas Kipf, **University of Amsterdam**

GNNs with **Edge** Embeddings

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)



Pros:

- Supports edge features
- More expressive than GCN
- As general as it gets (?)
- Supports sparse matrix ops

Formally:

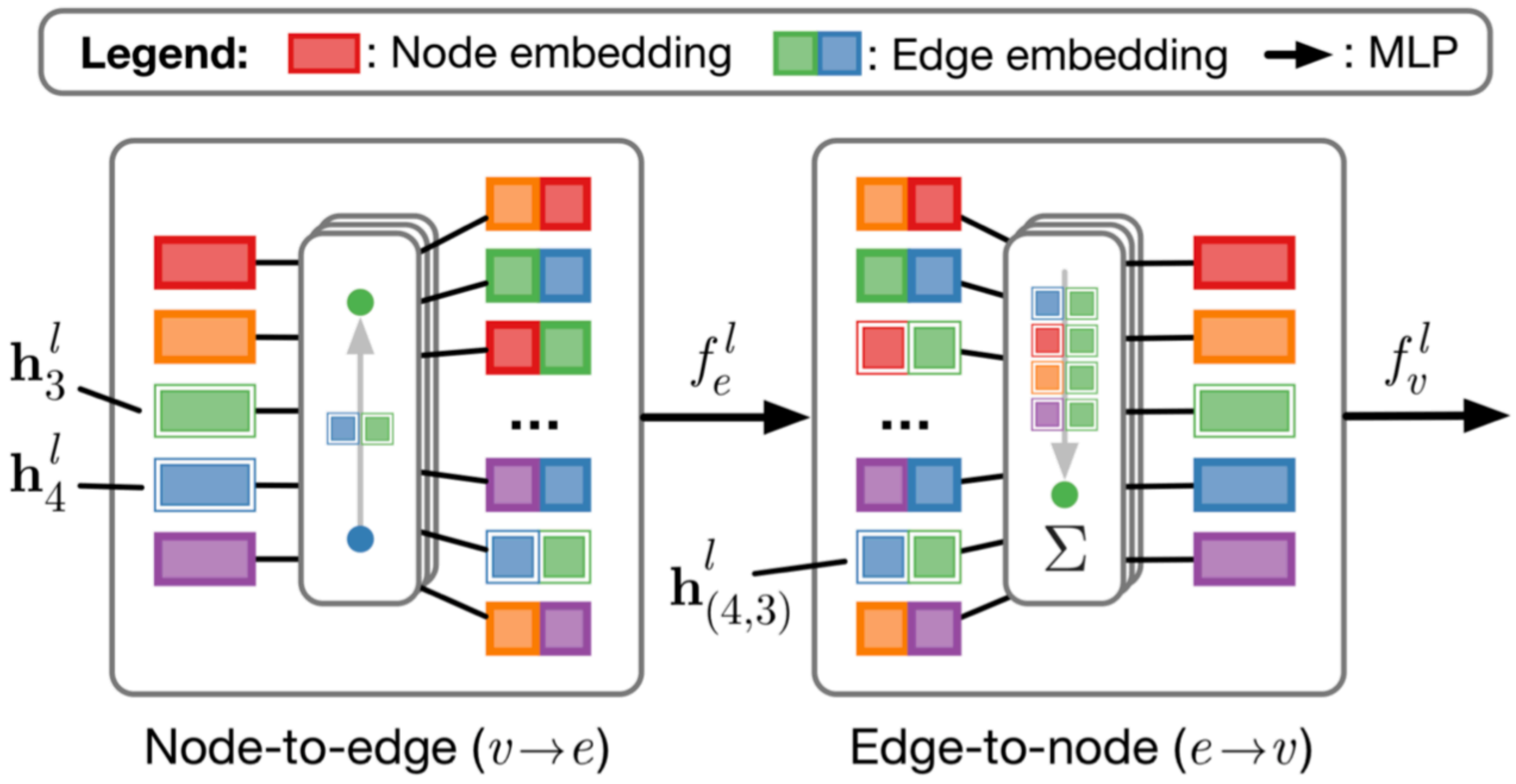
$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

* slide from Thomas Kipf, **University of Amsterdam**

GNNs with **Edge** Embeddings

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)



Pros:

- Supports edge features
- More expressive than GCN
- As general as it gets (?)
- Supports sparse matrix ops

Cons:

- Need to store intermediate edge-based activations
- Difficult to implement with subsampling

➡ In practice limited to small graphs

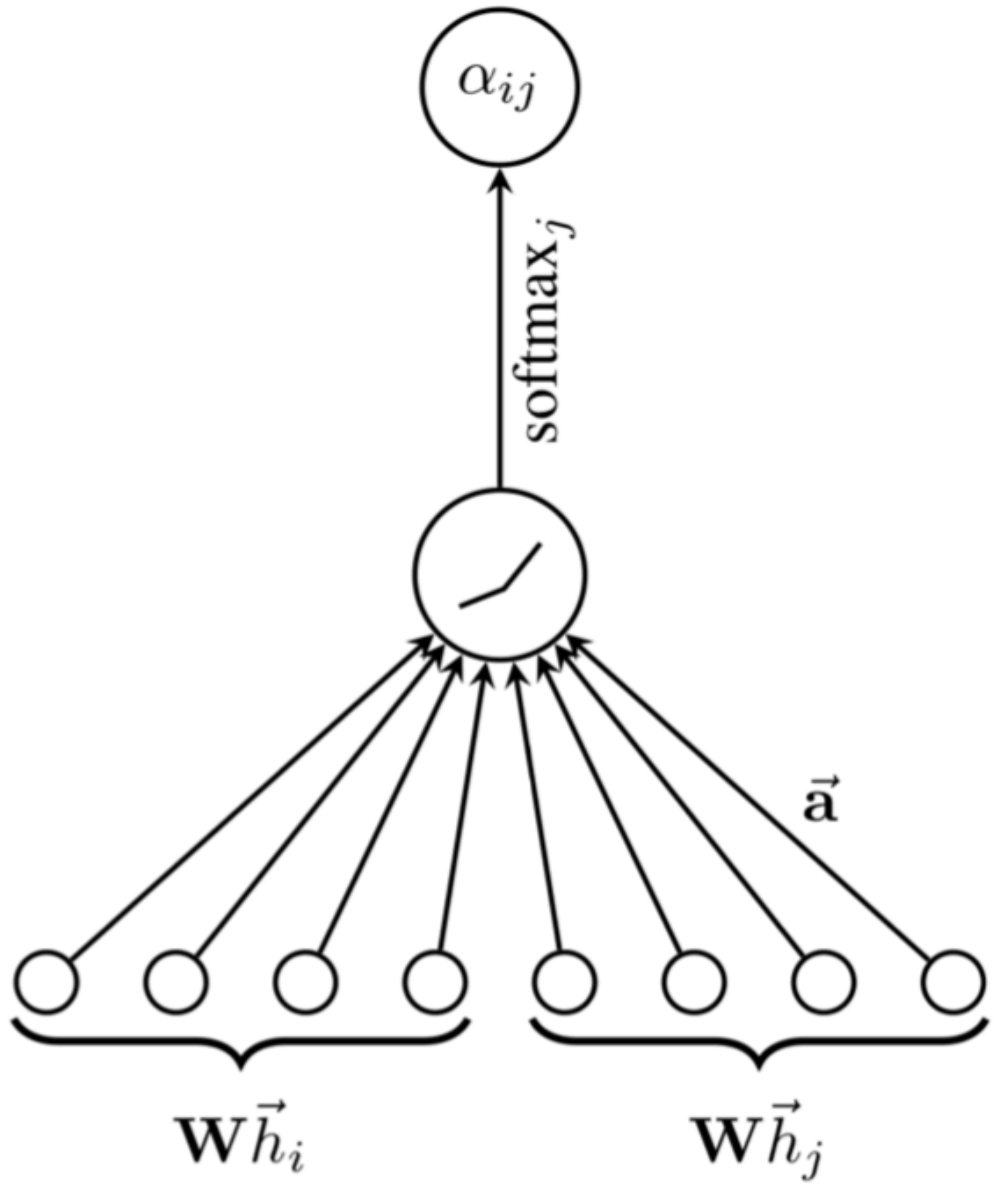
Formally:

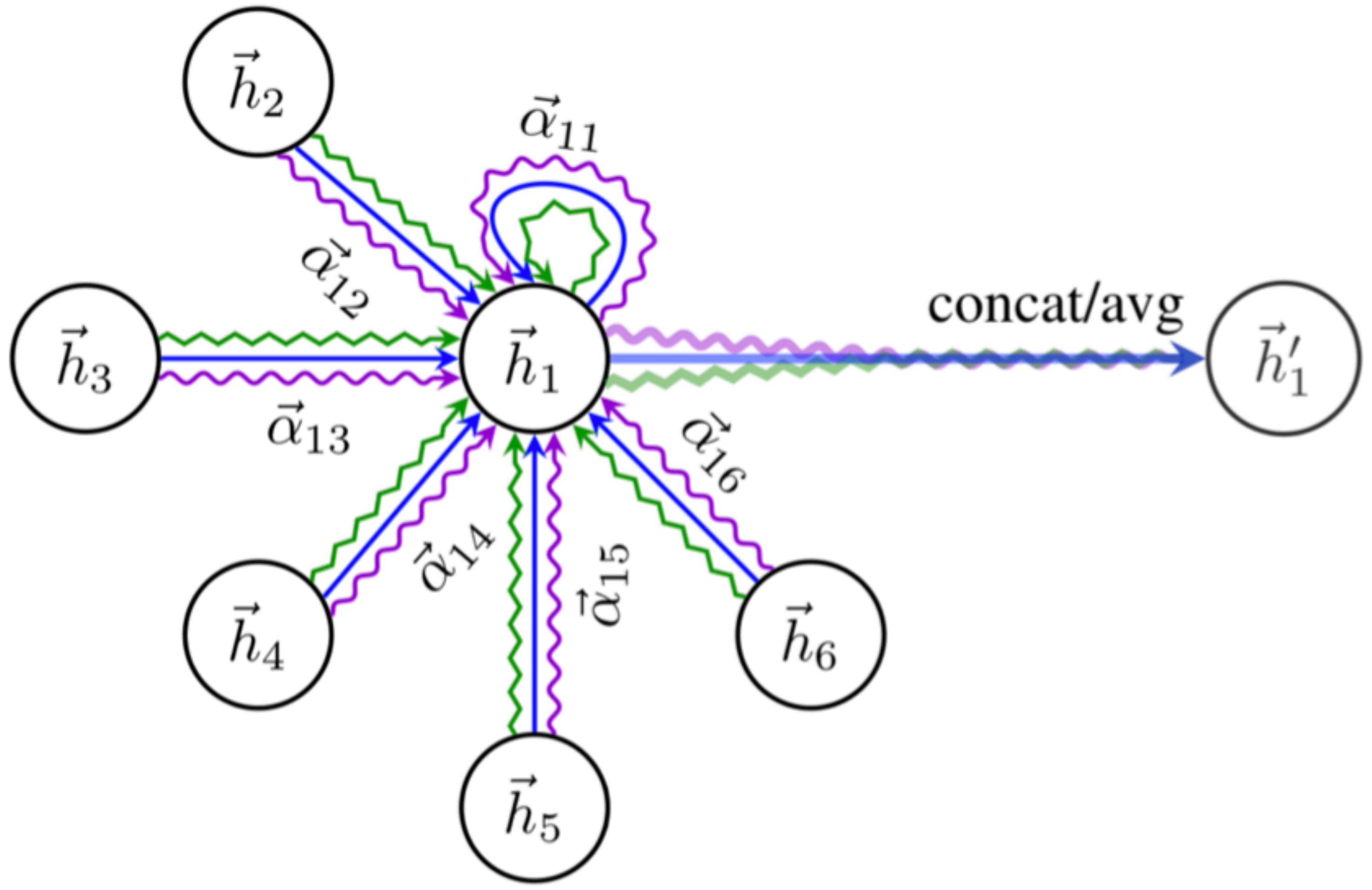
$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

Graph Neural Networks (GNNs) with **Attention**

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličković et al. (ICLR 2018)



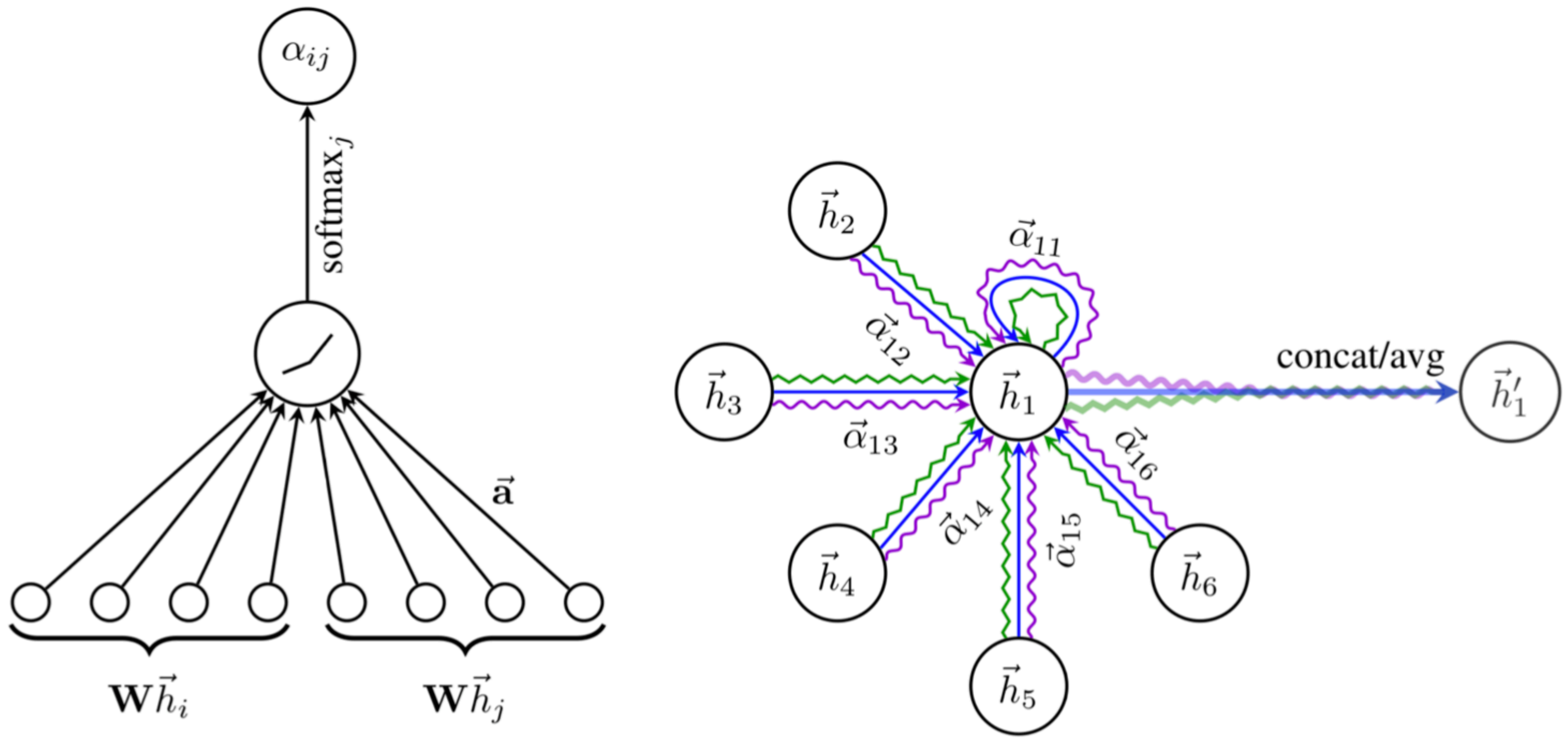


[Figure from Veličković et al. (ICLR 2018)]

$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

Graph Neural Networks (GNNs) with **Attention**

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličković et al. (ICLR 2018)



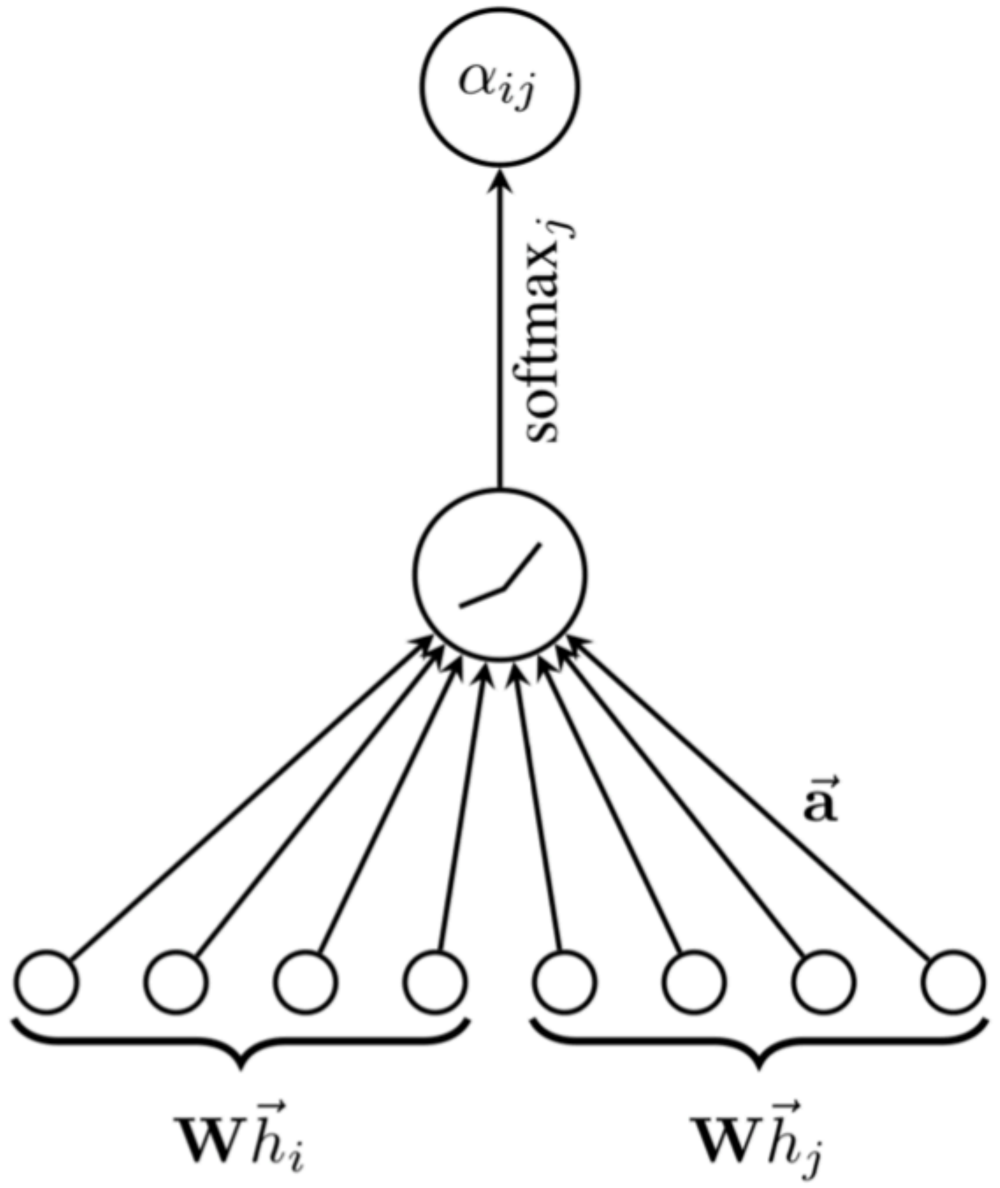
[Figure from Veličković et al. (ICLR 2018)]

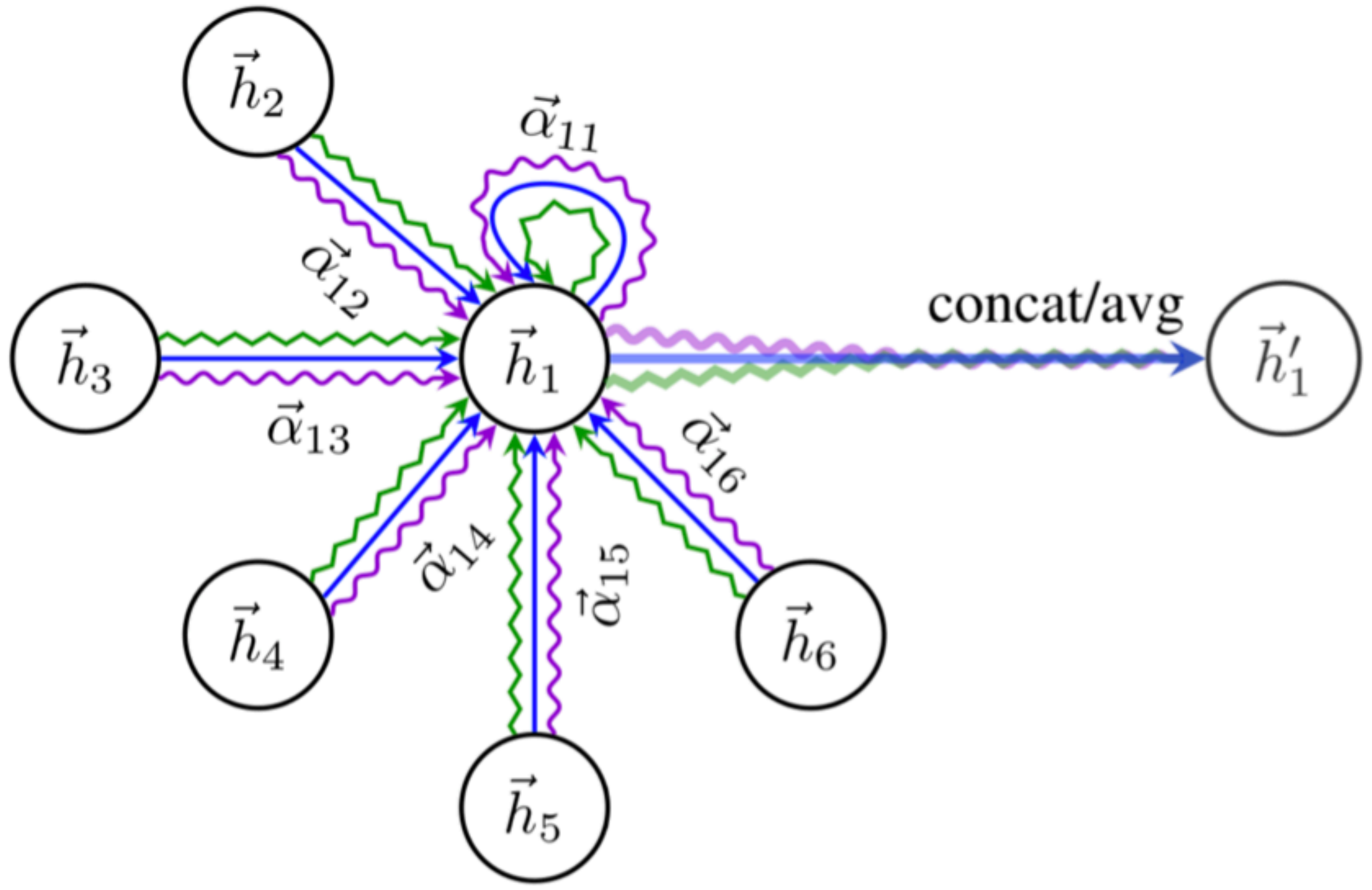
$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right) \qquad \alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

* slide from Thomas Kipf, **University of Amsterdam**

Graph Neural Networks (GNNs) with **Attention**

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličković et al. (ICLR 2018)





[Figure from Veličković et al. (ICLR 2018)]

Pros:

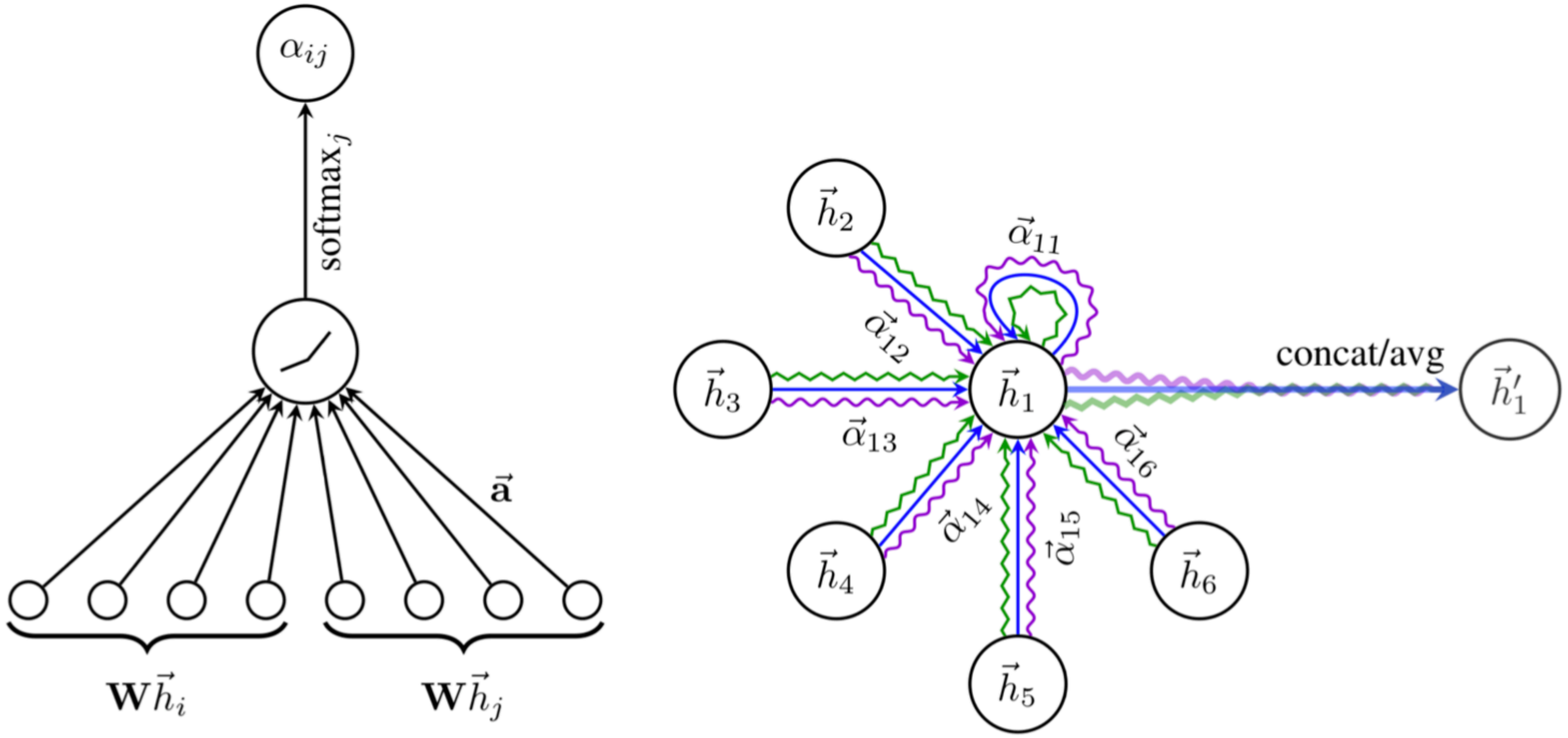
- No need to store intermediate edge-based activation vectors (when using dot-product attn.)
- Slower than GCNs but faster than GNNs with edge embeddings

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right) \qquad \alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

* slide from Thomas Kipf, **University of Amsterdam**

Graph Neural Networks (GNNs) with **Attention**

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličković et al. (ICLR 2018)



[Figure from Veličković et al. (ICLR 2018)]

Pros:

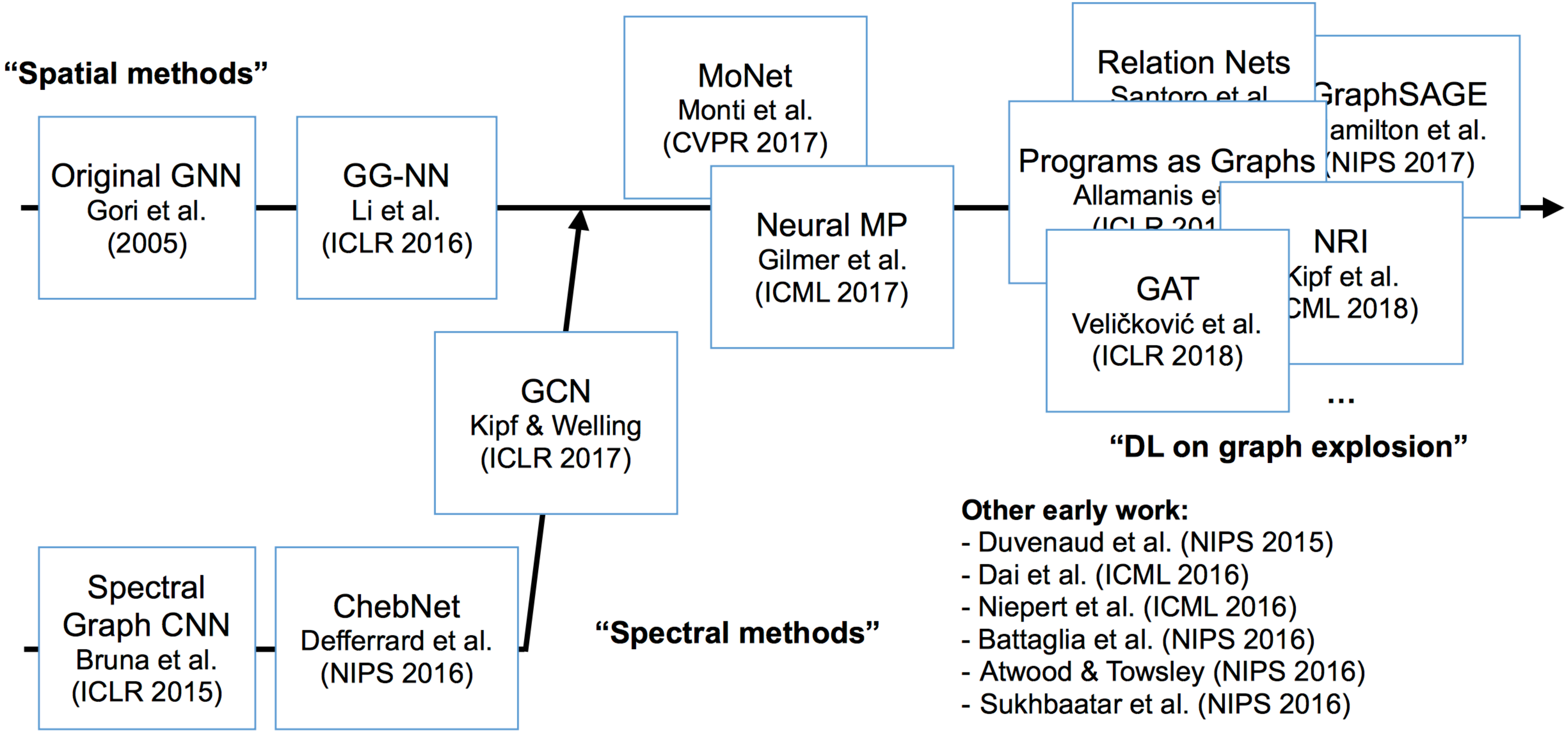
- No need to store intermediate edge-based activation vectors (when using dot-product attn.)
- Slower than GCNs but faster than GNNs with edge embeddings

Cons:

- (Most likely) less expressive than GNNs with edge embeddings
- Can be more difficult to optimize

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right) \qquad \alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

A **Brief History** of Graph Neural Nets



Other early work:
- Duvenaud et al. (NIPS 2015)
- Dai et al. (ICML 2016)
- Niepert et al. (ICML 2016)
- Battaglia et al. (NIPS 2016)
- Atwood & Towsley (NIPS 2016)
- Sukhbaatar et al. (NIPS 2016)

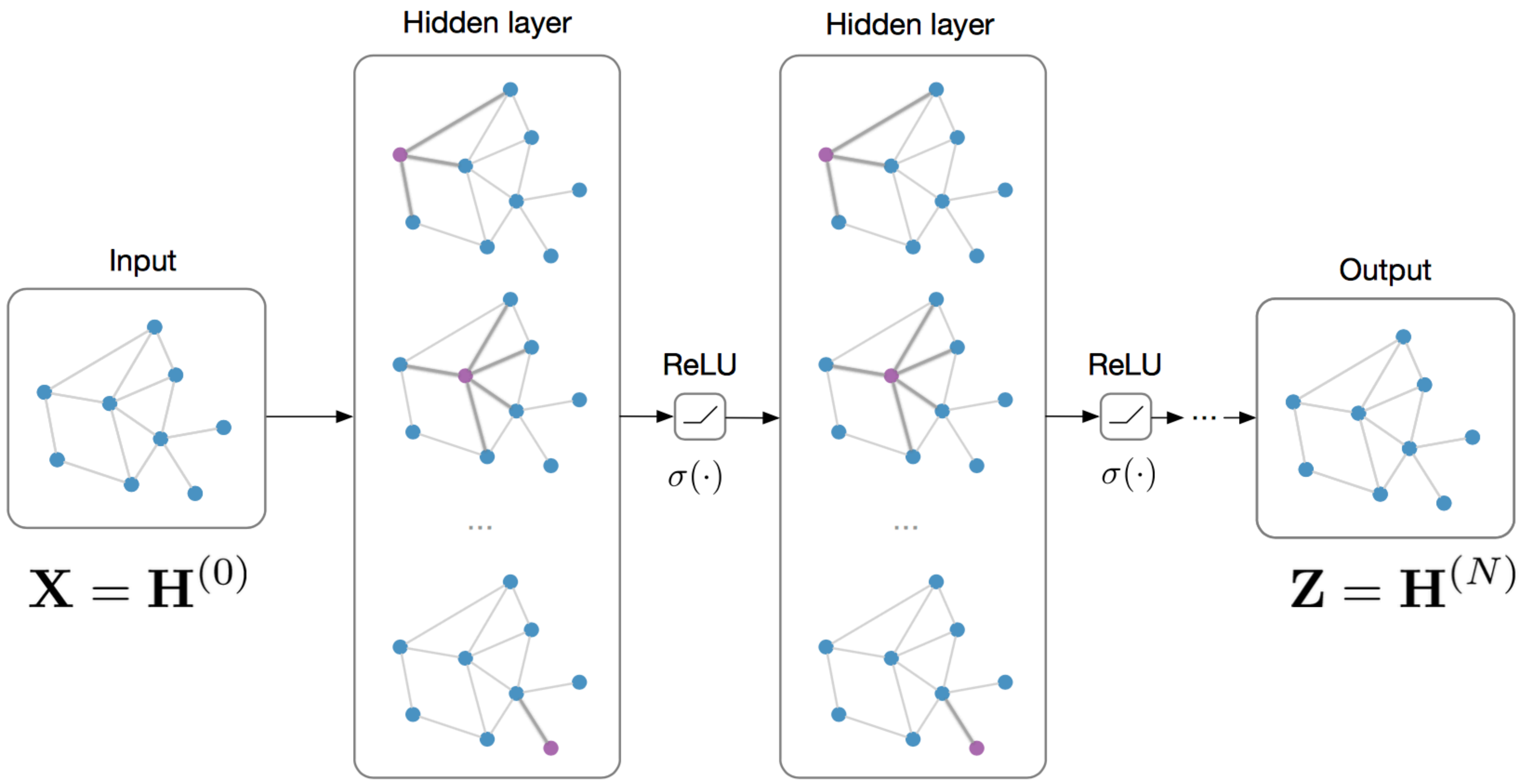
(slide inspired by Alexander Gaunt's talk on GNNs)

* slide from Thomas Kipf, **University of Amsterdam**

How do we use GNN / GCN for real problems?

Classification and **Link Prediction** with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$

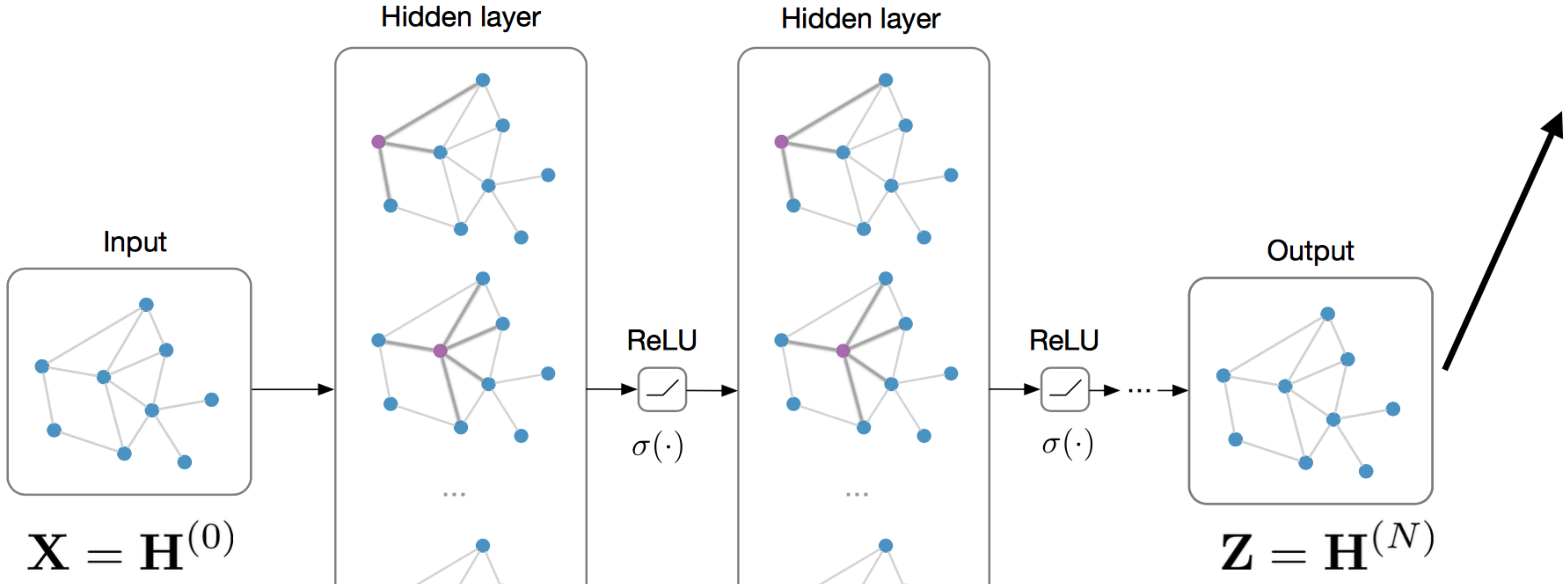


$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

* slide from Thomas Kipf, **University of Amsterdam**

Classification and **Link Prediction** with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Node classification:

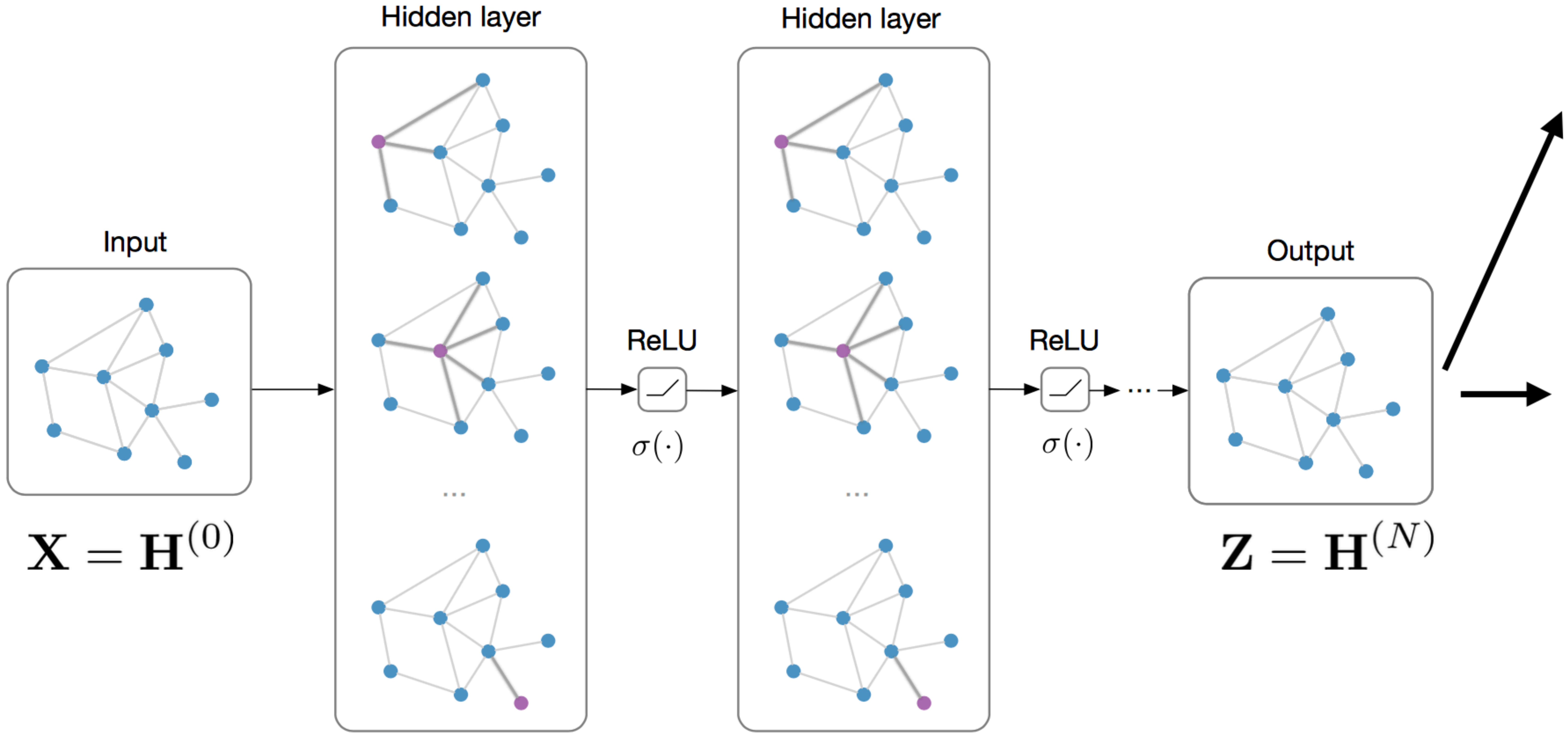
$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

* slide from Thomas Kipf, **University of Amsterdam**

Classification and **Link Prediction** with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Node classification:

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

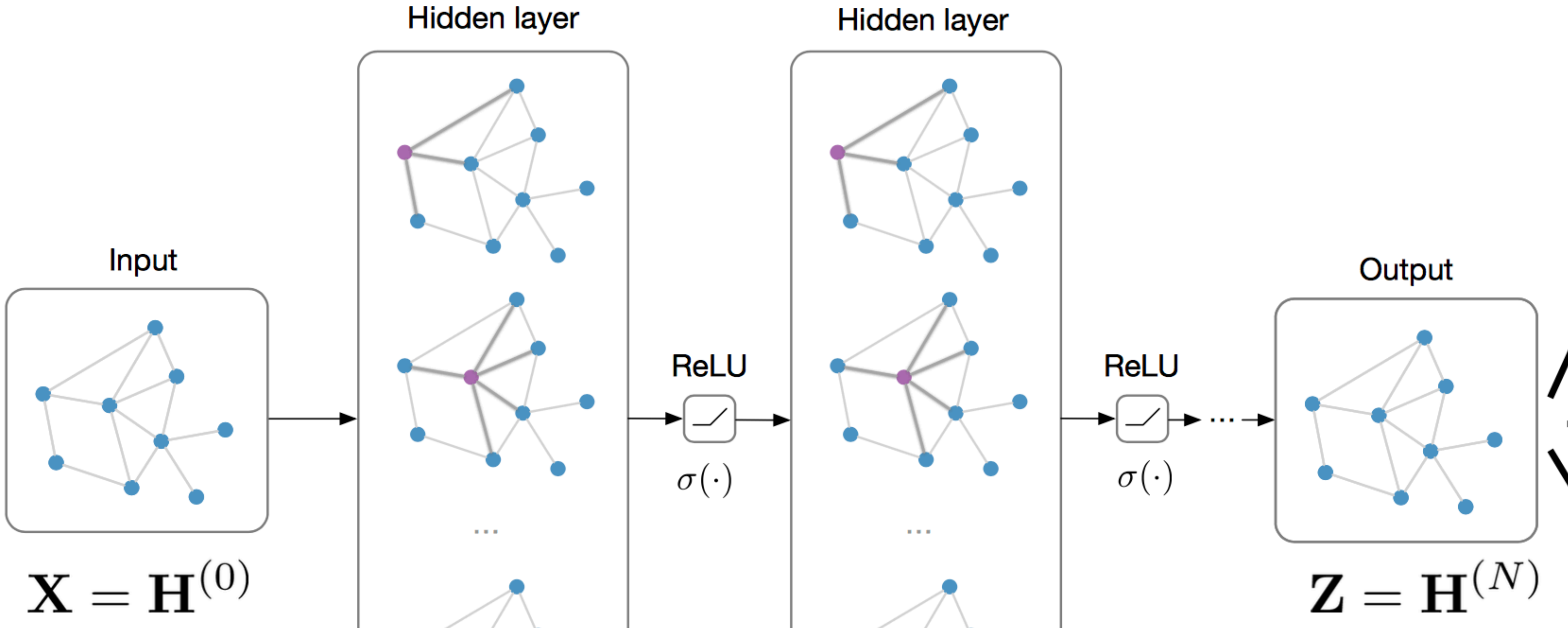
$$\mathrm{softmax}(\textstyle\sum_n \mathbf{z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

* slide from Thomas Kipf, **University of Amsterdam**

Classification and **Link Prediction** with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Node classification:

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\mathrm{softmax}(\textstyle\sum_n \mathbf{z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

$$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

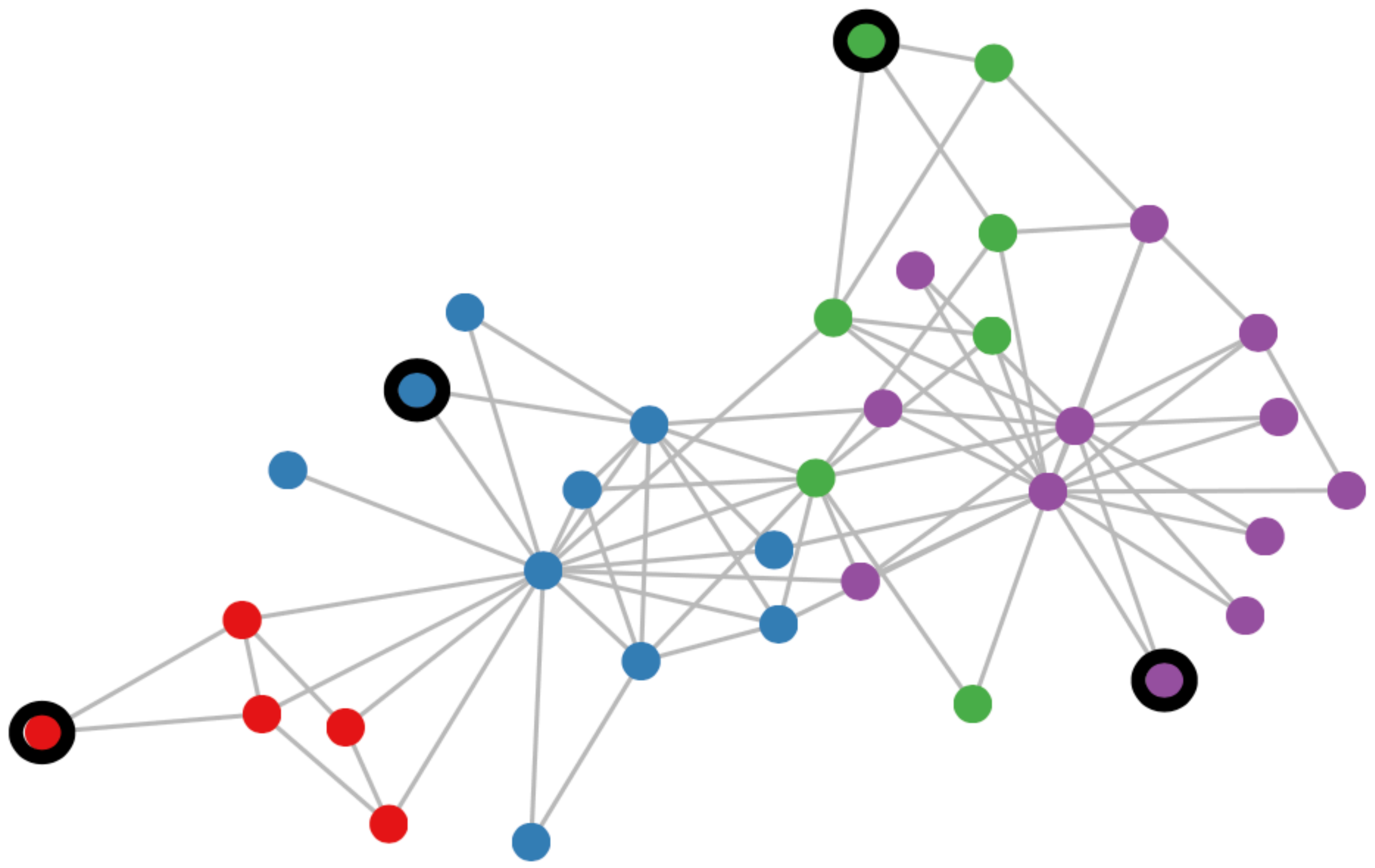
* slide from Thomas Kipf, **University of Amsterdam**

Semi-supervised Classification on Graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:
Predict node label of unlabeled nodes



* slide from Thomas Kipf, **University of Amsterdam**

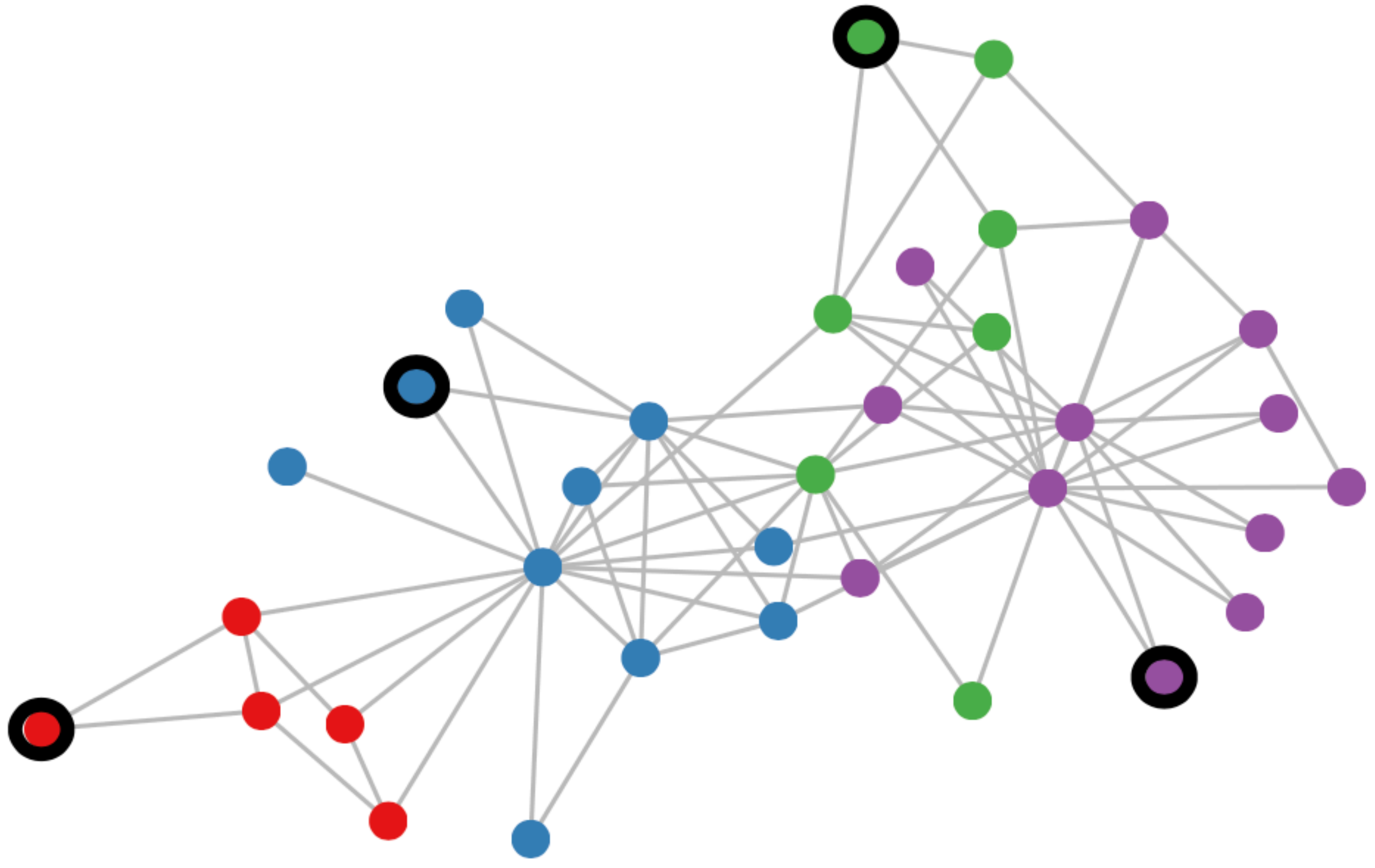
Semi-supervised Classification on Graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:

Predict node label of unlabeled nodes



Evaluate loss on labeled nodes only:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

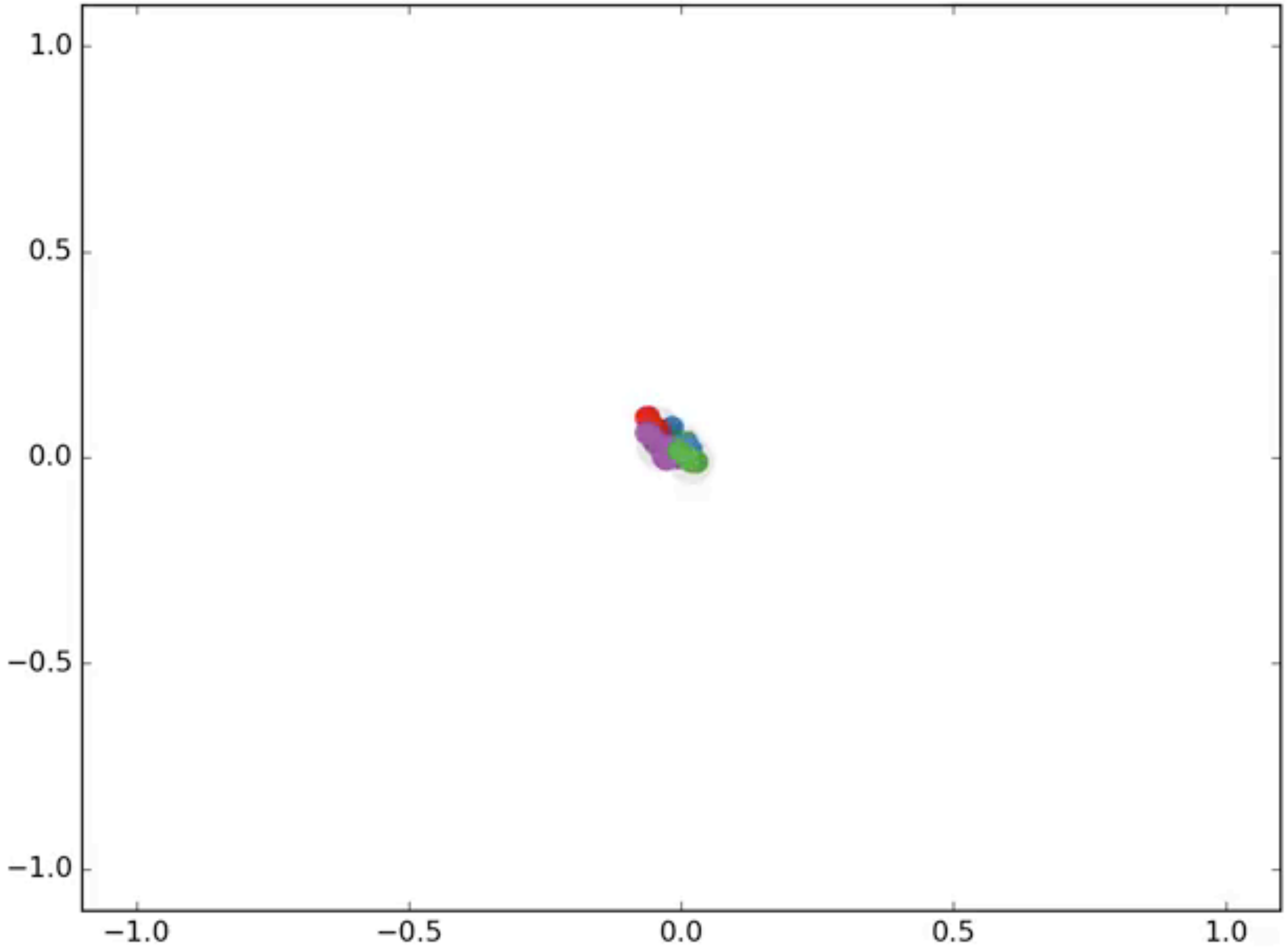
$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

* slide from Thomas Kipf, **University of Amsterdam**

Semi-supervised Classification on Graphs



* slide from Thomas Kipf, **University of Amsterdam**

Conclusions

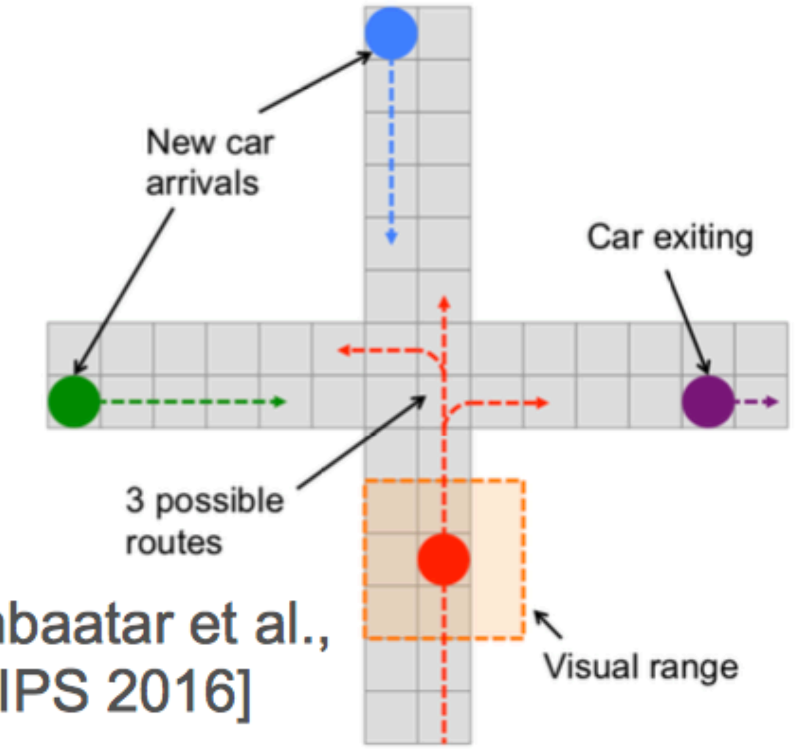
— **Deep learning on graphs works and is very effective!**

— Exciting area: lots of new applications and extensions (hard to keep up)

Relational reasoning

[Santoro et al., NIPS 2017]

Multi-Agent RL



[Sukhbaatar et al., NIPS 2016]

GCN for recommendation on 16 billion edge graph!

[Leskovec lab, Stanford]

Open problems:

— Theory

— Scalable, stable generative models

— Learning on large, evolving data

— Multi-modal and cross-model learning (e.g., sequence2graph)

* slide from Thomas Kipf, **University of Amsterdam**

Graph Neural Nets (GNNs) are strict Generalizations of Traditional Neural Nets

(CNNs / RNNs can be implemented using GNNs / GCNs, but this is inefficient)