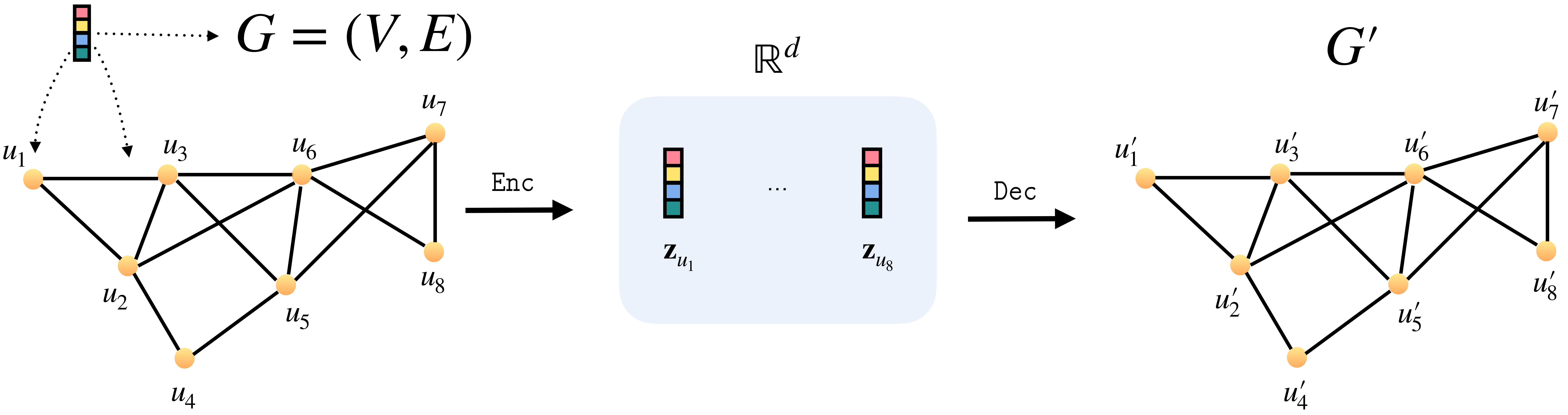
Lecture 4: Message Passing Neural Network Architectures

Relational Learning

İsmail İlkan Ceylan Advanced Topics in Machine Learning, University of Oxford 24.01.2022

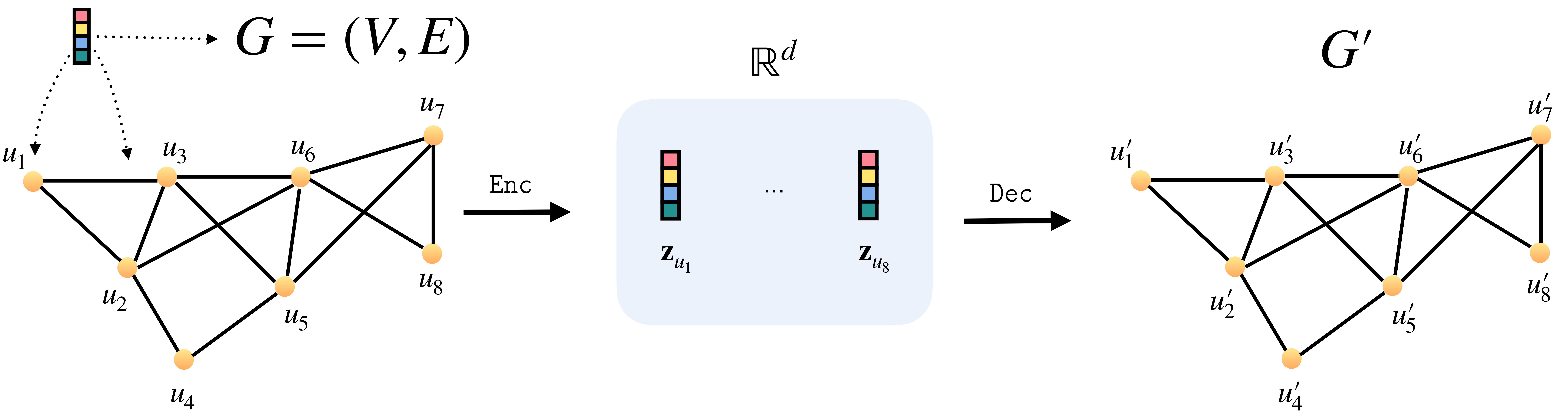
Encoder-Decoder



Knowledge Graph Embeddings

Lecture 1-2

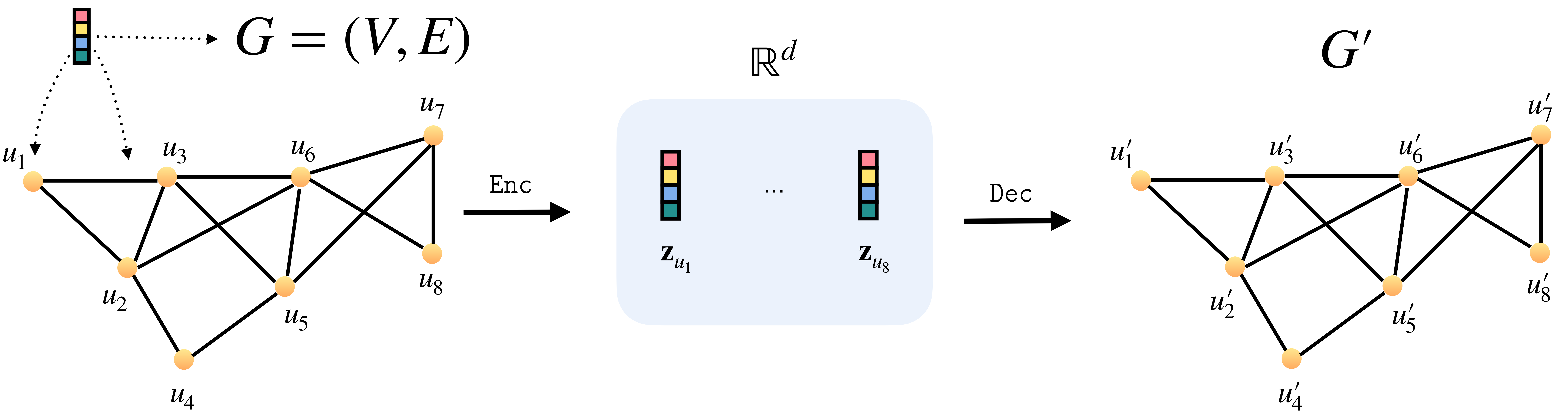
Encoder-Decoder



MPNNs (Lecture 3)

$$\mathbf{h}_u^{(t)} = \textit{combine}^{(t)}\left(\mathbf{h}_u^{(t-1)}, \textit{aggregate}^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right)\right)$$

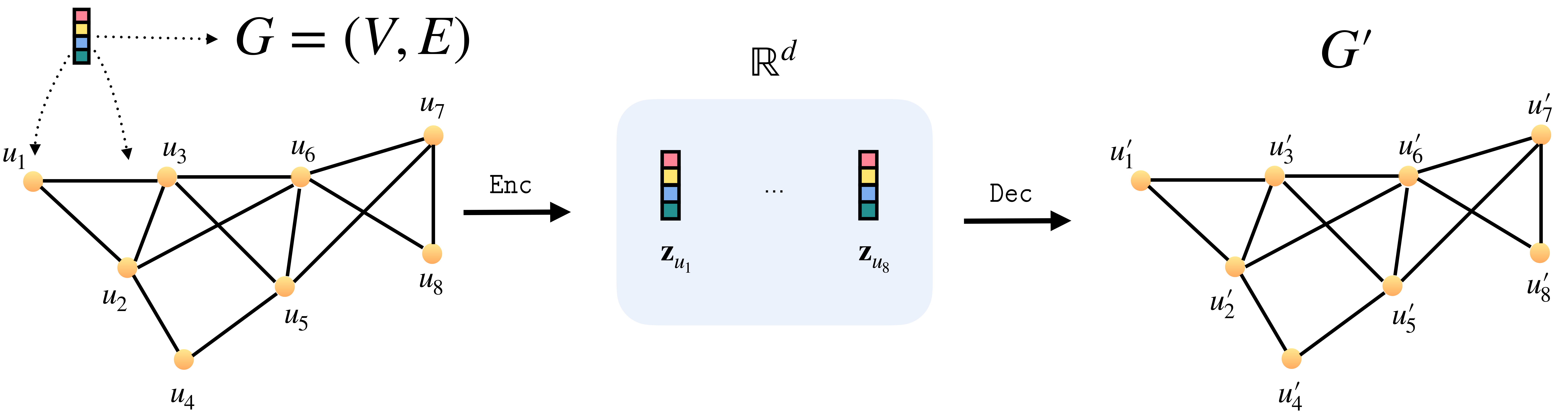
Encoder-Decoder



Base GNN Model (Lecture 3)

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

Encoder-Decoder



Today's Lecture

Popular GNN models

Overview

- Historical perspectives for graph neural network models
- Gated graph neural networks
- Graph convolutional networks
- Graph attention networks
- Graph isomorphism networks
- Relational message passing architectures
- Limitations of MPNNs: over-smoothing, over-squashing, inexpressiveness
- Summary

Historical Perspectives for Graph Neural Networks

From convolutions to graph convolutions:

Motivated by the success of convolutional neural networks: generalize Euclidean convolutions to the graph domain (Bruna et al., 2014) - Graph convolutional networks (Kipf and Welling, 2016).

From graph isomorphism testing to graph representation learning:

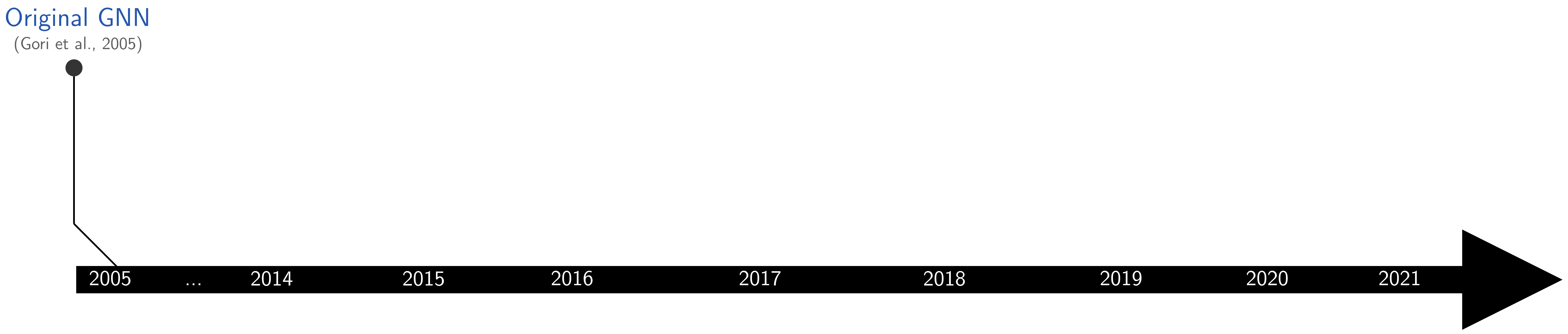
Learning over graphs requires to distinguish graphs: MPNNs cannot distinguish all graphs, and so they have limited expressive power. The connection to graph isomorphism testing offers many theoretical insights.

From belief propagation to MPNNs:

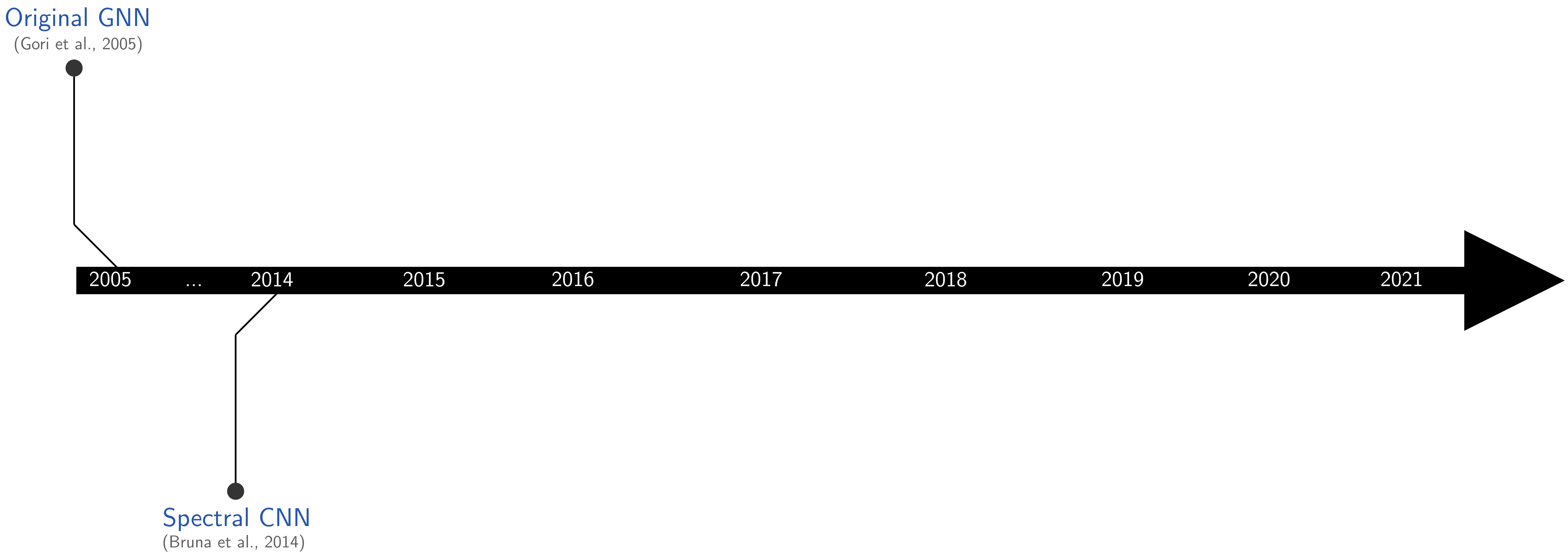
Message passing is used in the context of probabilistic graphical models (i.e., belief propagation (Pearl, 82)). Dai et al., (2016): Neural message passing algorithms are analogues of certain message passing algorithms common in variational inference to infer distributions over latent variables.

Graph Neural Networks

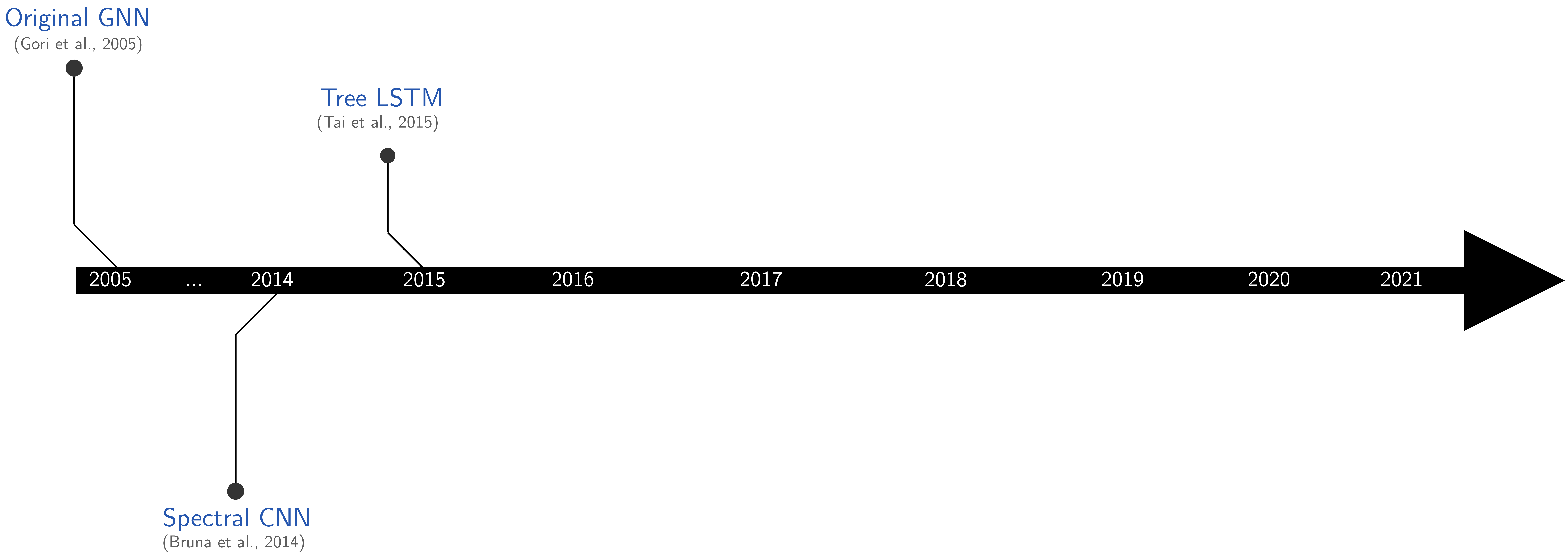
Graph Neural Networks



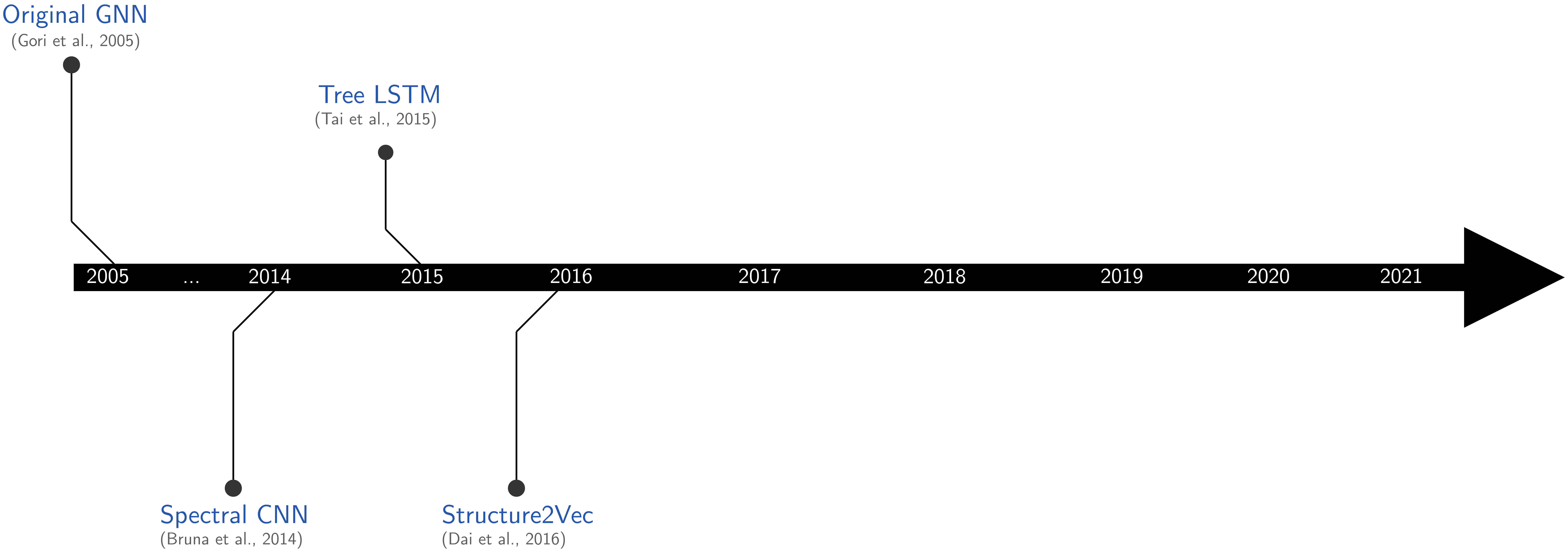
Graph Neural Networks



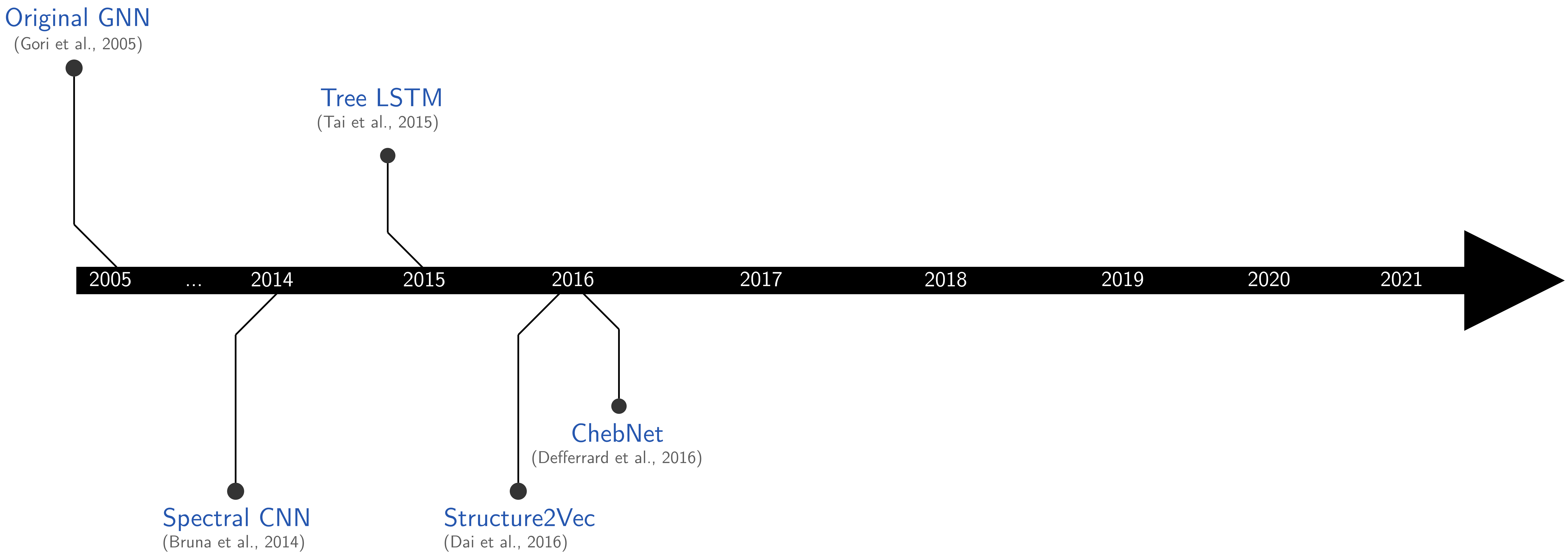
Graph Neural Networks



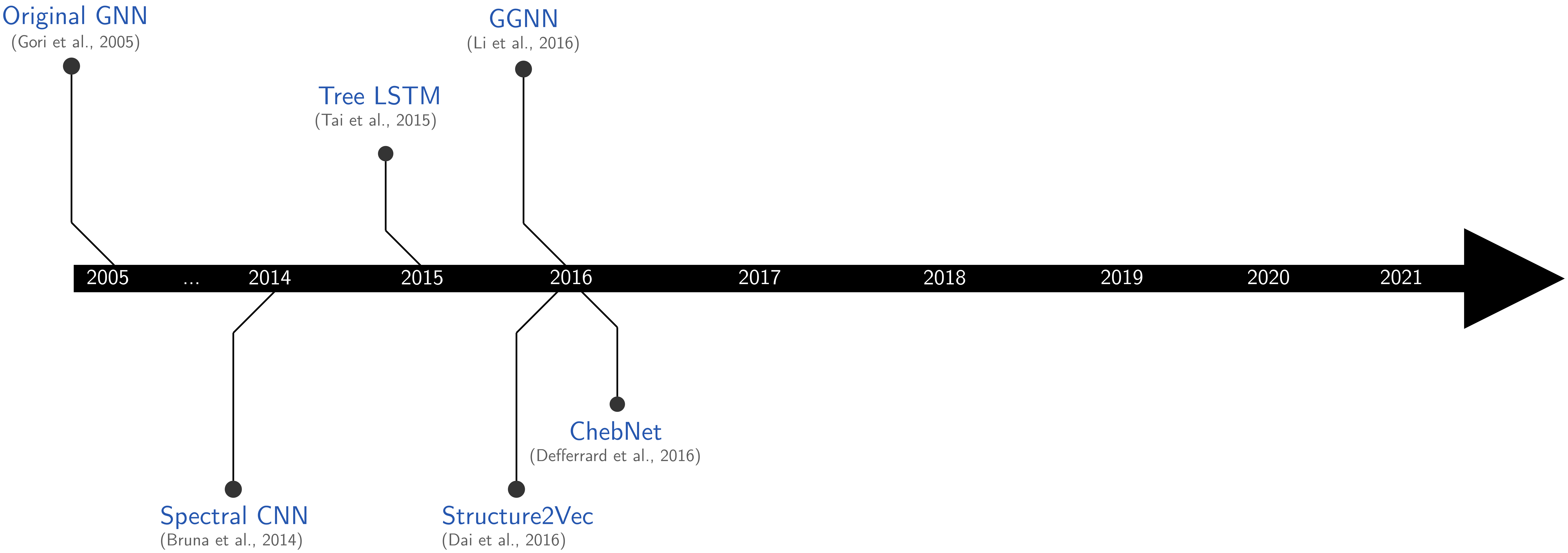
Graph Neural Networks



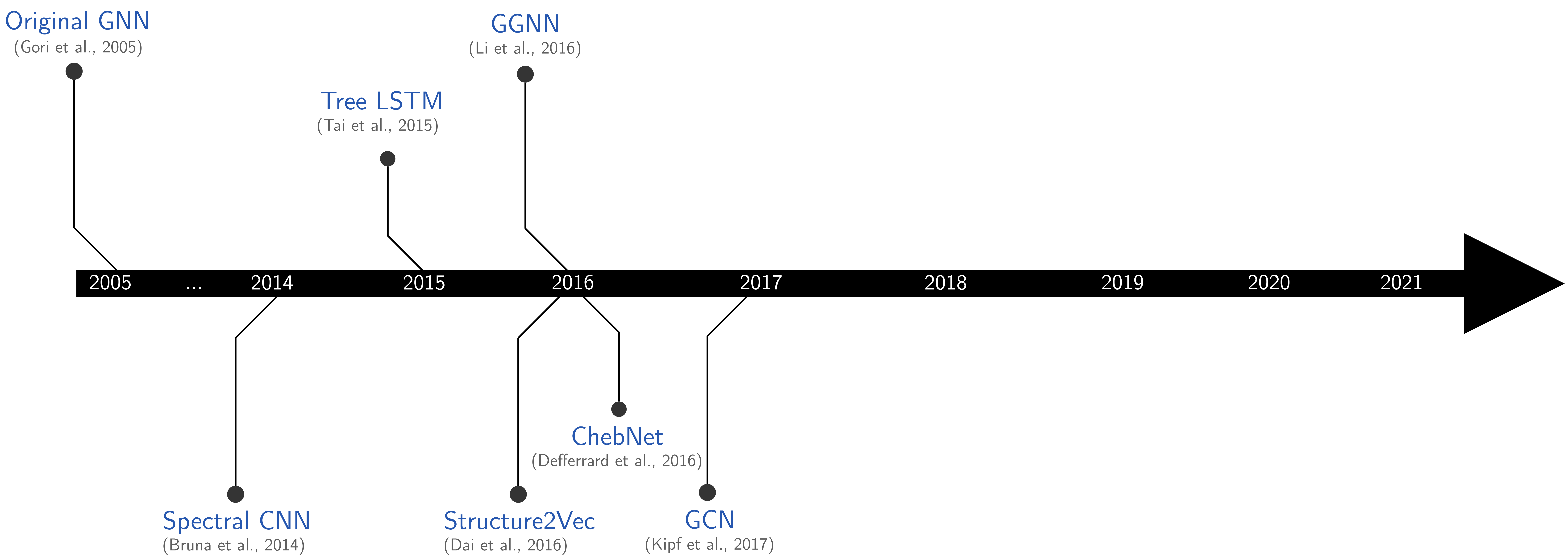
Graph Neural Networks



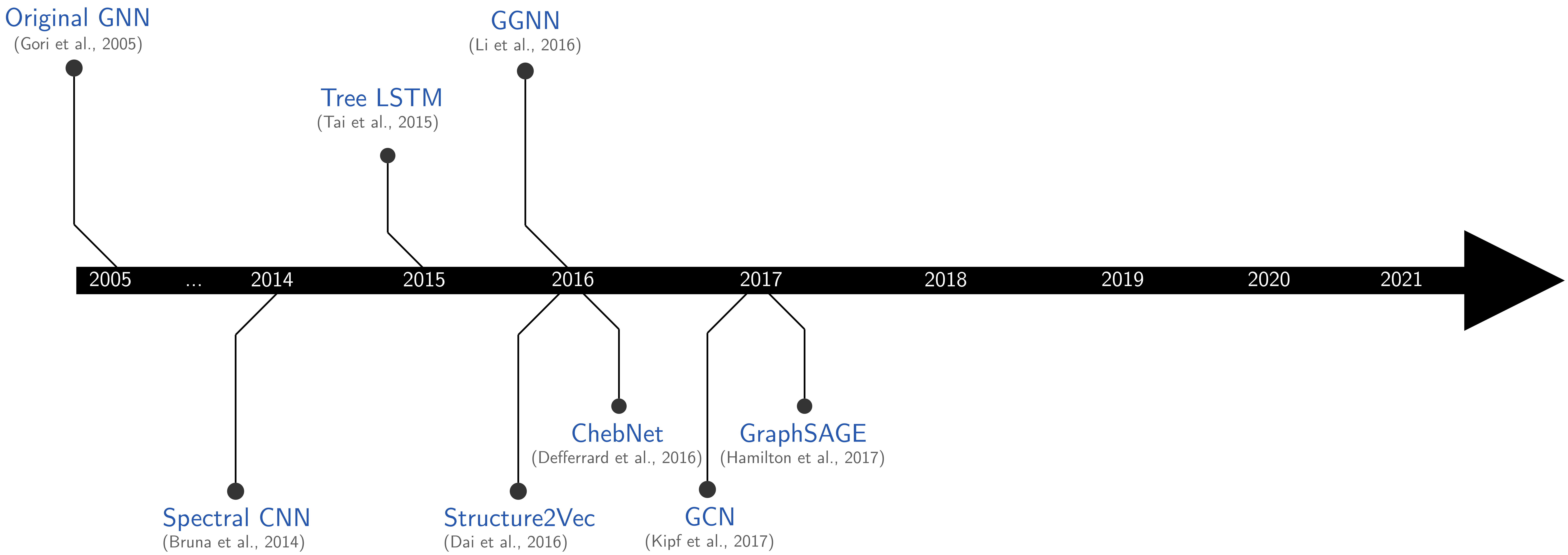
Graph Neural Networks



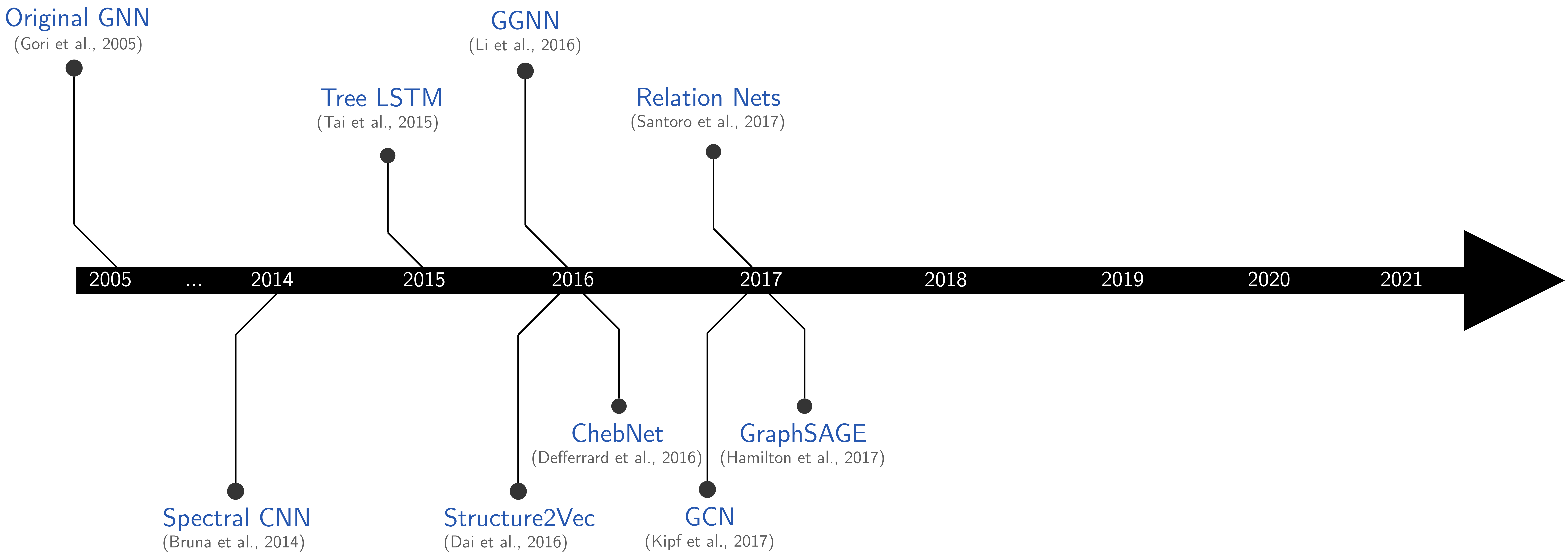
Graph Neural Networks



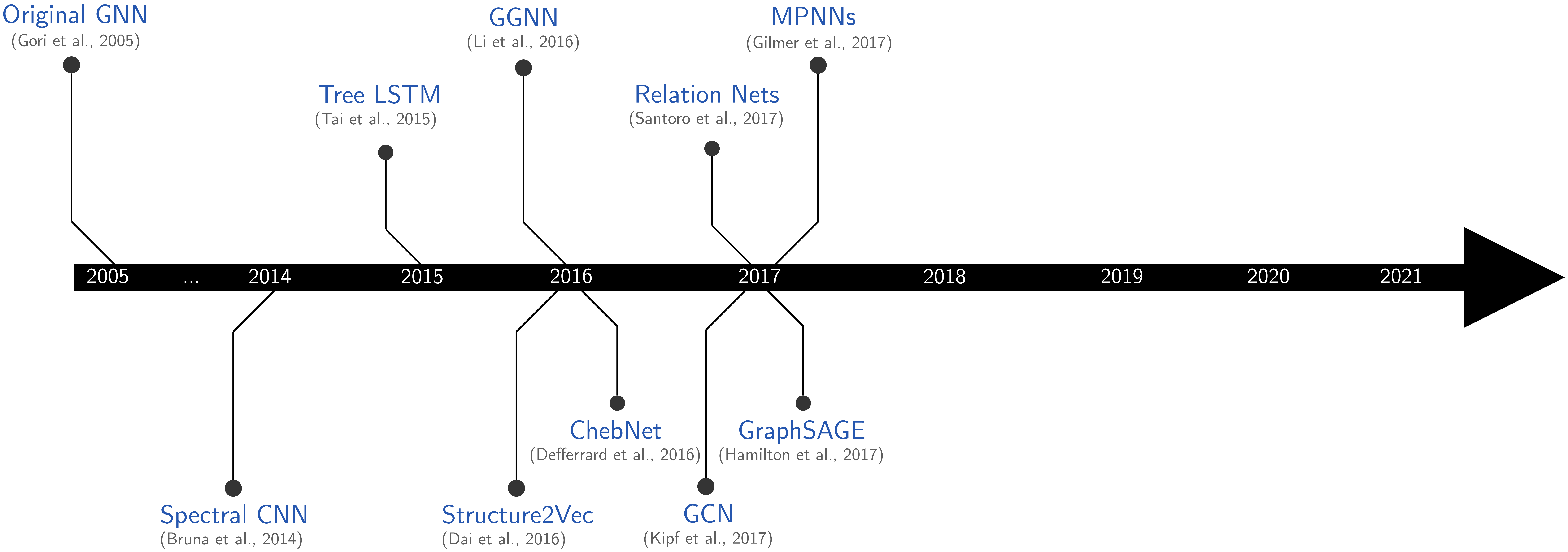
Graph Neural Networks



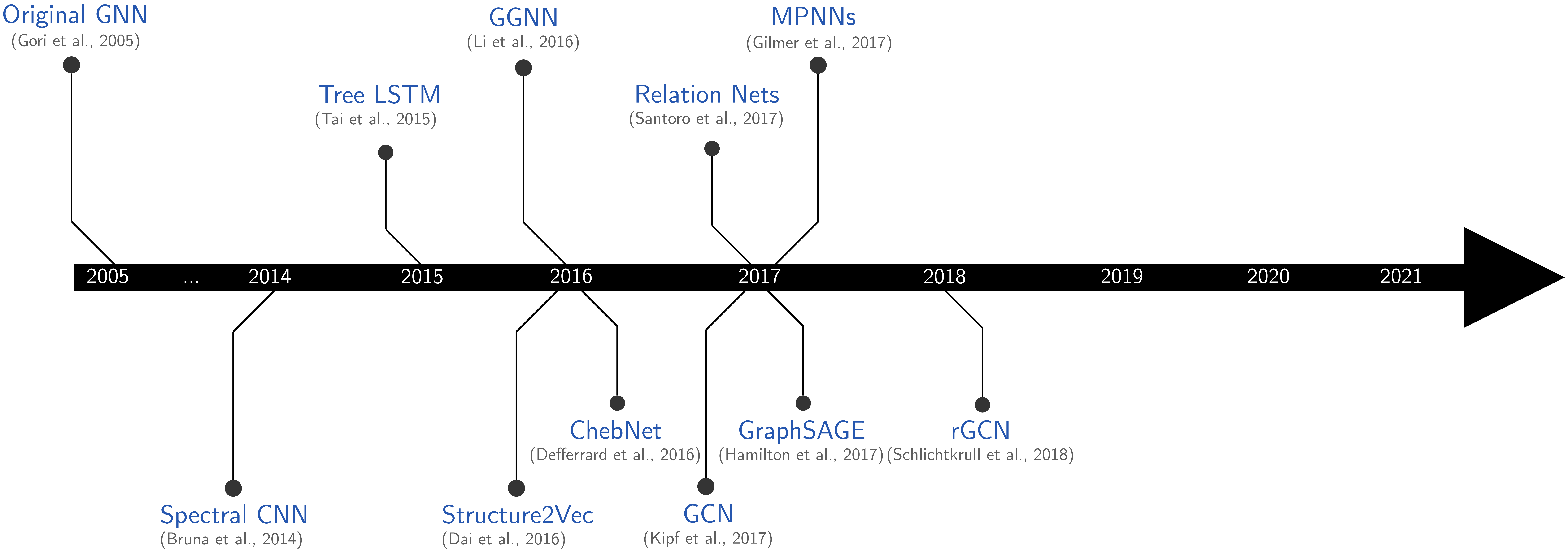
Graph Neural Networks



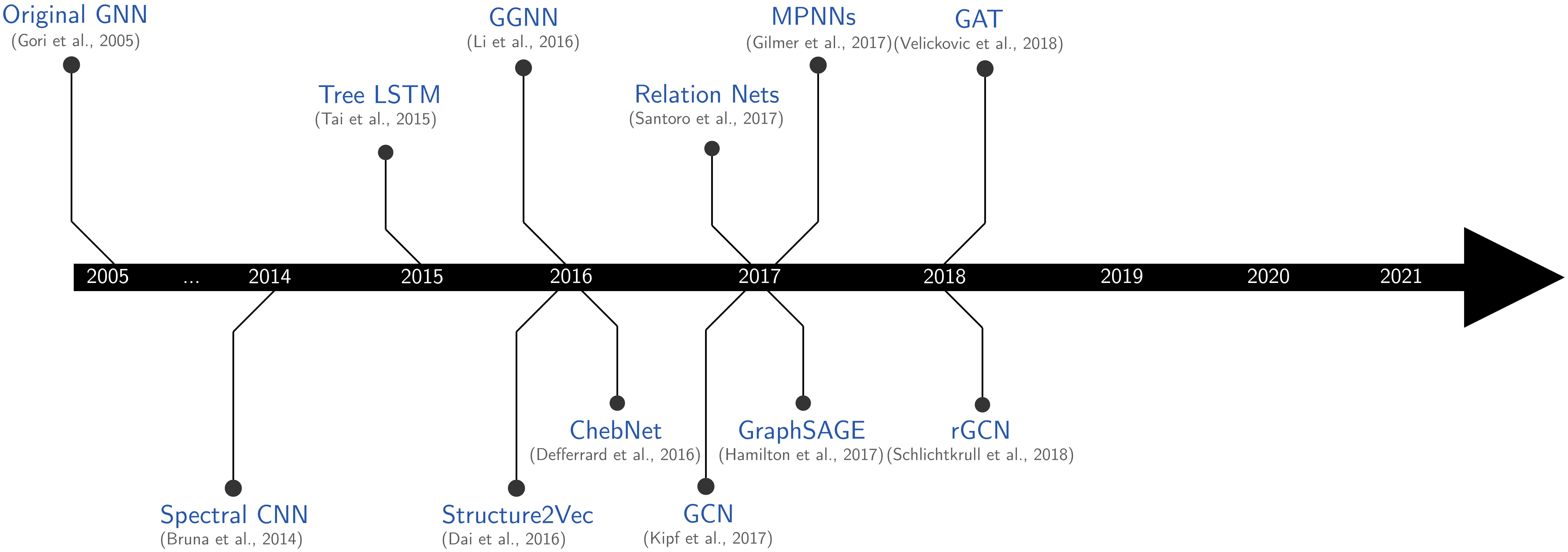
Graph Neural Networks



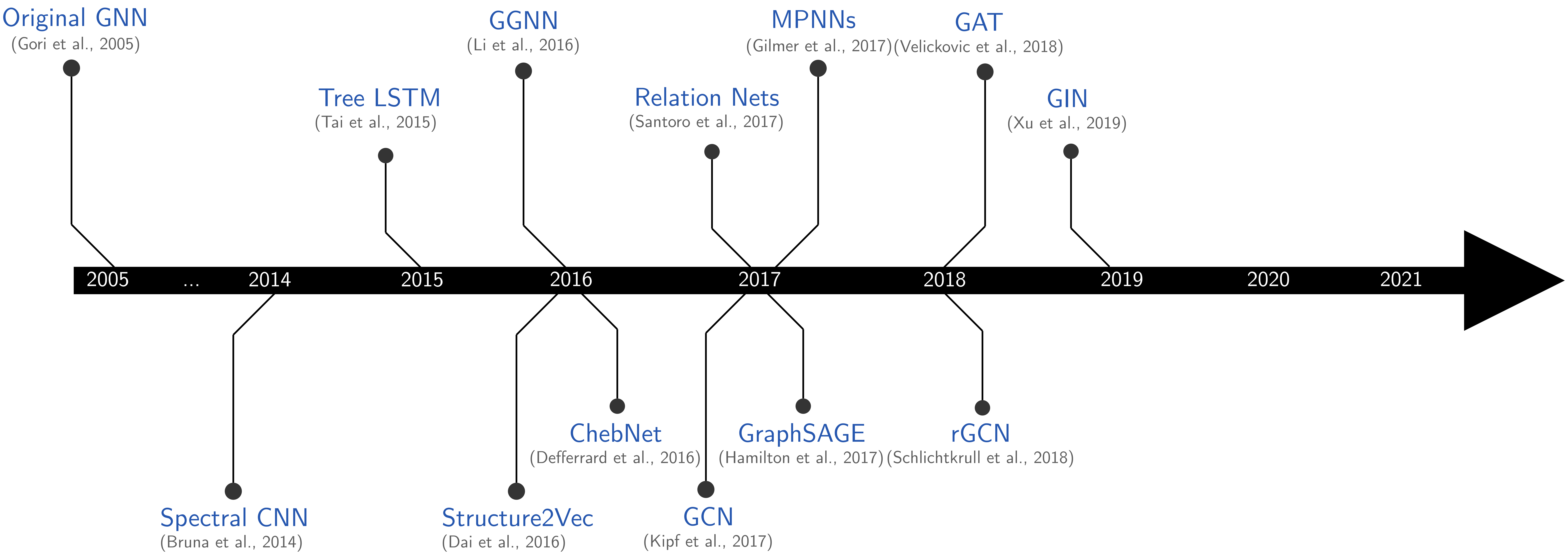
Graph Neural Networks



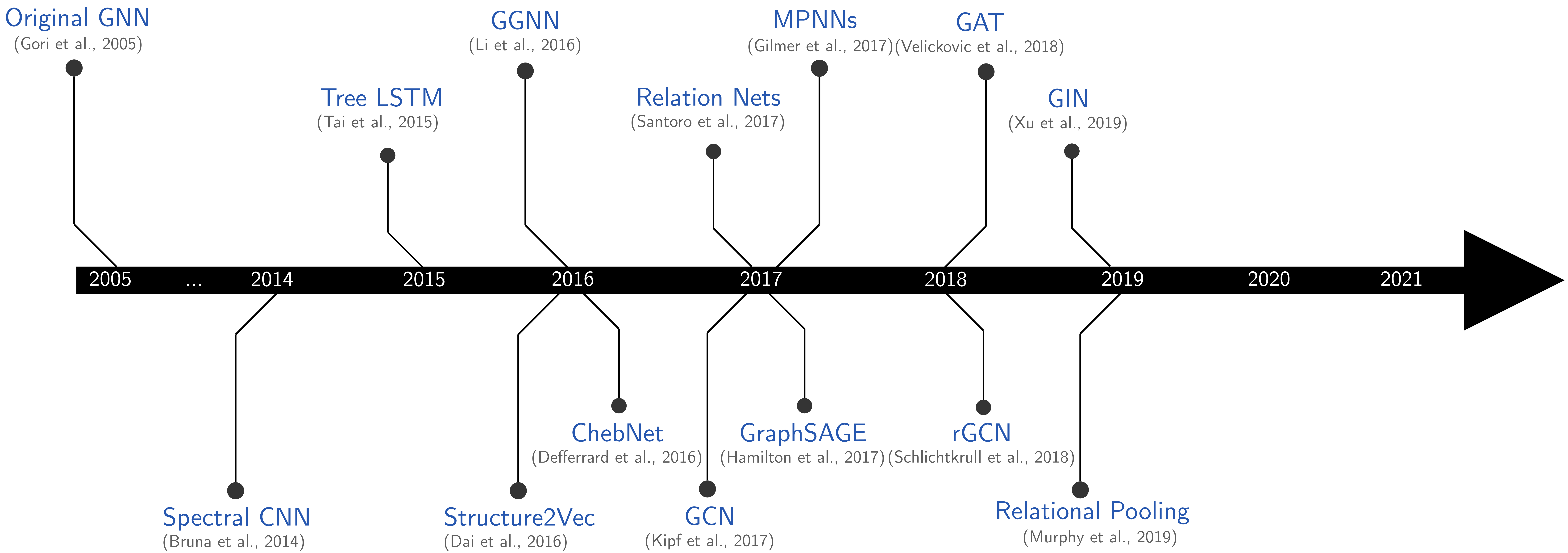
Graph Neural Networks



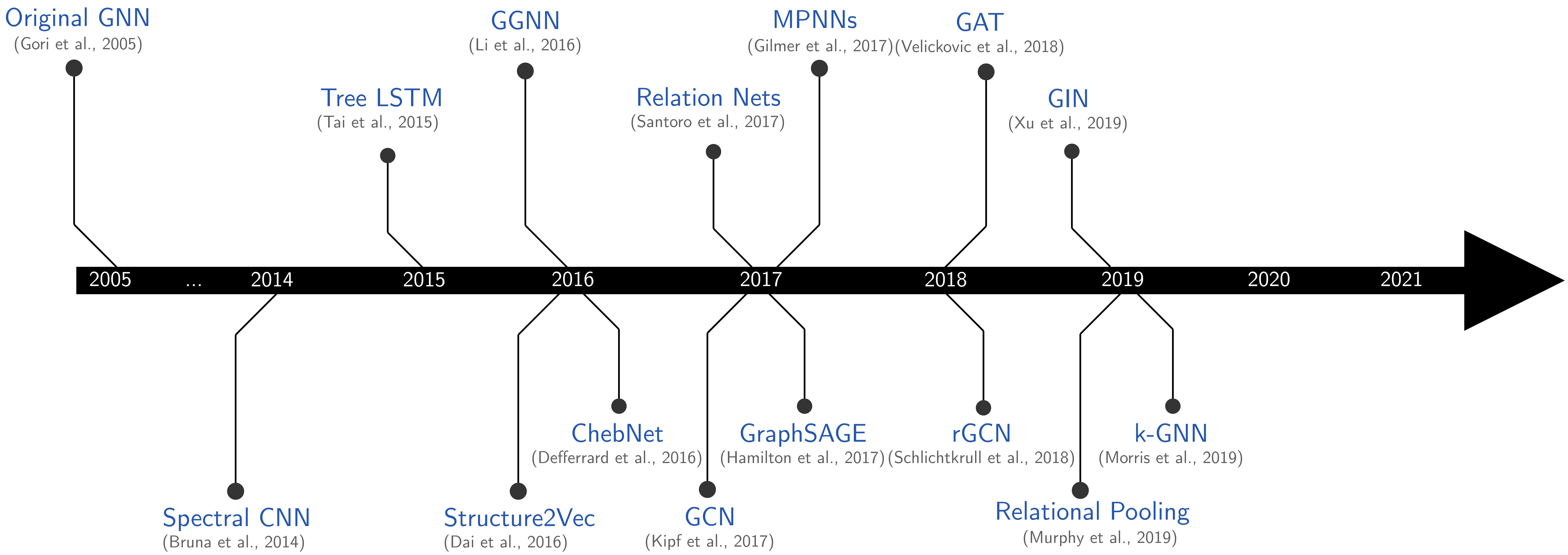
Graph Neural Networks



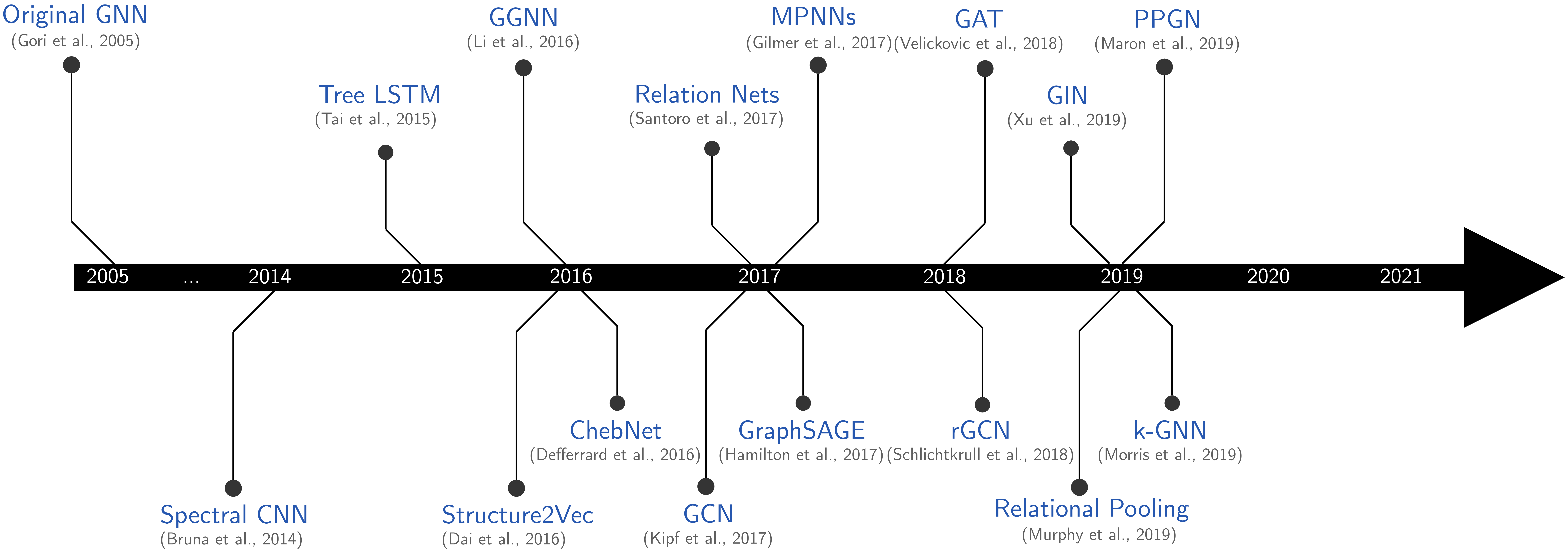
Graph Neural Networks



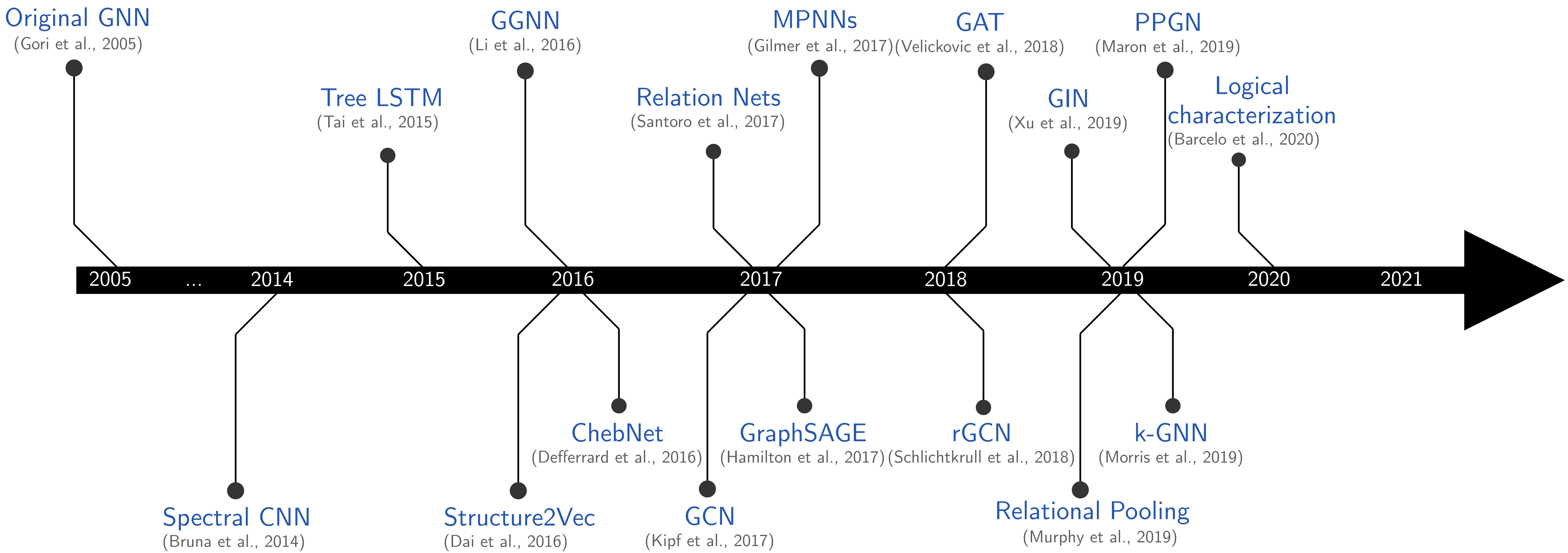
Graph Neural Networks



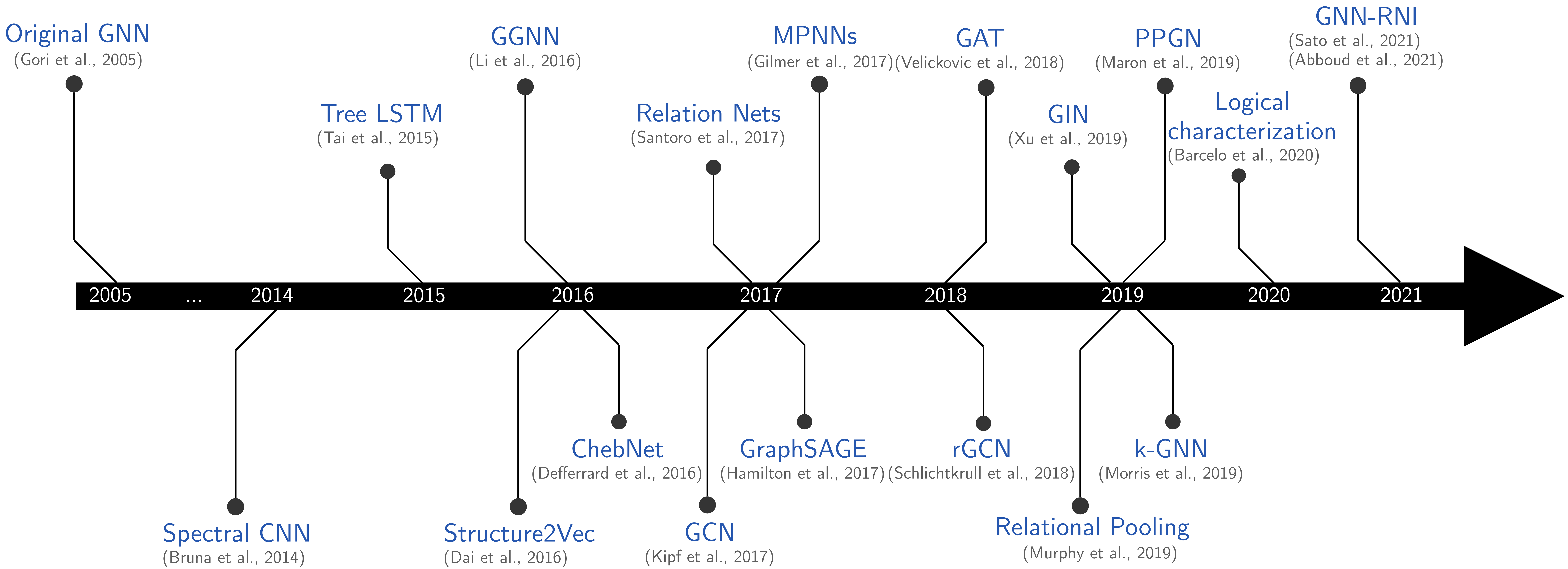
Graph Neural Networks



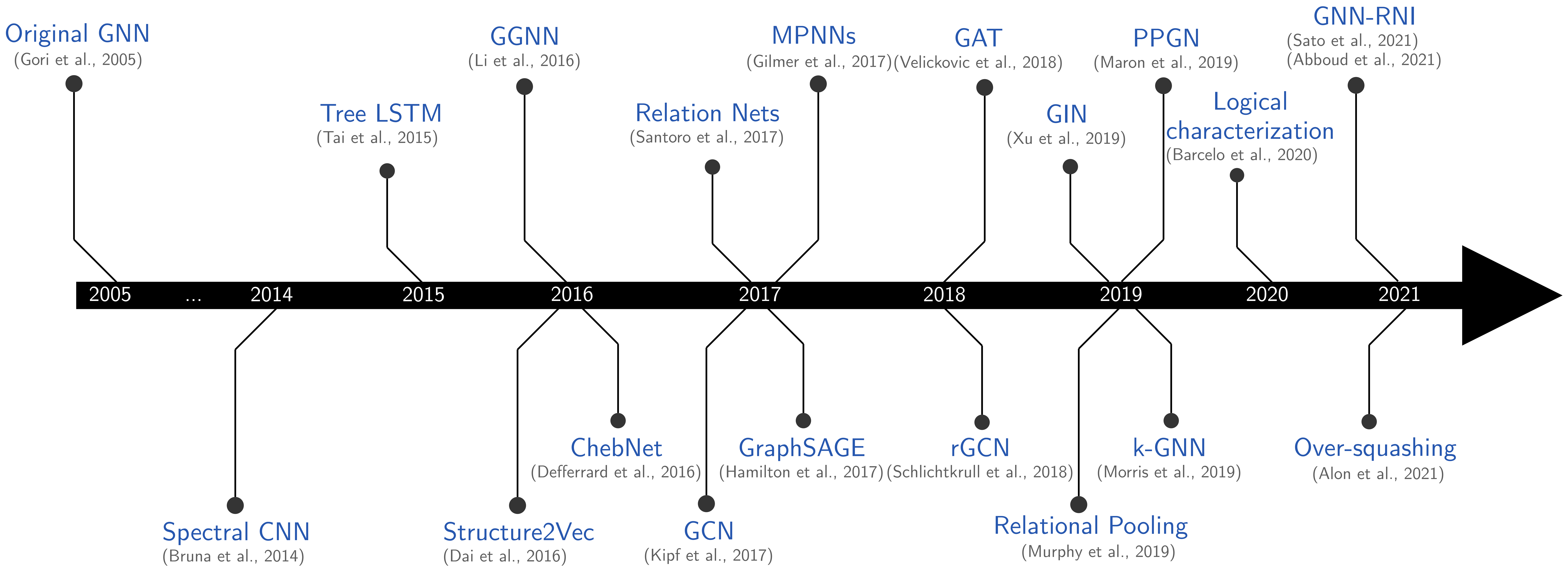
Graph Neural Networks



Graph Neural Networks



Graph Neural Networks



Gated Graph Neural Networks

Node Embeddings as a Sequence

MPNNs employ an iterative algorithm to learn node embeddings:

$$\mathbf{h}_u^{(t)} = \mathit{combine}^{(t)}\Big(\mathbf{h}_u^{(t-1)}, \mathit{aggregate}^{(t)}\big(\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\}\big)\Big)$$

Message passing can be seen as a sequential process:

- Every node has an initial state characterized by the node features $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.
- Every node's state is updated after each message passing iteration based on:
  - Previous state of the node
  - States of the neighboring nodes
- This process terminates at the end of message passing, yielding final states.

Sequence Modeling: Refresher

Spam detection: Identify whether an email is spam or not.

- Sentences processed word by word by neural sequence models (e.g., GRU) and the state is updated based on
  - the most recent word,
  - a state which stores information about earlier words
- This process is repeated until we see each word, yields a final representation for the overall sentence.

Idea: Maintain a state in memory, and based on the new state and input, decide to retain or update your state:

$$\mathbf{R}^t = \sigma(\mathbf{X}^t\mathbf{W}_{xr} + \mathbf{H}^{(t-1)}\mathbf{W}_{hr} + \mathbf{b}_r) \qquad \mathbf{Z}^t = \sigma(\mathbf{X}^t\mathbf{W}_{xz} + \mathbf{H}^{(t-1)}\mathbf{W}_{hz} + \mathbf{b}_z)$$

$$\tilde{\mathbf{H}}^t = \tanh(\mathbf{X}^t\mathbf{W}_{xh} + \left(\mathbf{R}^t \odot \mathbf{H}^{(t-1)}\right)\mathbf{W}_{hh} + \mathbf{b}_h) \qquad \mathbf{H}^t = \mathbf{Z}^t \odot \mathbf{H}^{t-1} + (1 - \mathbf{Z}^t) \odot \tilde{\mathbf{H}}^t$$

Gated Graph Neural Networks

Using the state abstraction for nodes in a graph, MPNNs can employ three separate computations:

1. **Message computation**: Based on a node's current state
2. **Message aggregation**: Node-level aggregation
3. **State update**: A recurrent unit takes the current state, the aggregation of messages, and updates.

Gated graph neural networks (Li et al., 2016), update the representation $\mathbf{h}_u$ for each node $u \in V$ as:

$$\mathbf{h}_u^{(t)} = GRU\left(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{W}^{(t)} \mathbf{h}_v^{(t-1)}\right)$$

Message computation via multiplication by a weight matrix, aggregate by sum, and combine with a GRU.

Graph Convolutional Networks

Graph Convolutional Networks

The base GCN model is an instance of the MPNN framework and defined as:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u) + N(v)}}\right)$$

The base MPNN model is very similar to the base MPNN with self-loops (modulo normalization):

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \mathbf{h}_v^{(t-1)}\right)$$

Question: Can we view this model as applying convolutions over graphs?

Idea: View each message as a signal and matrix transformations applying to the signals as convolutions.

Revisiting the Basic Model

The base MPNN model is defined as a node-level equation:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

The base MPNN model can be written as a graph-level equation:

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{H}^{(t-1)}\mathbf{W}_{self}^{(t)} + \mathbf{A}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\right),$$

...where the matrix $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ has the node representations at layer t.

Revisiting the Basic Model

The base MPNN model is defined as a node-level equation:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

The base MPNN model can be written as a graph-level equation:

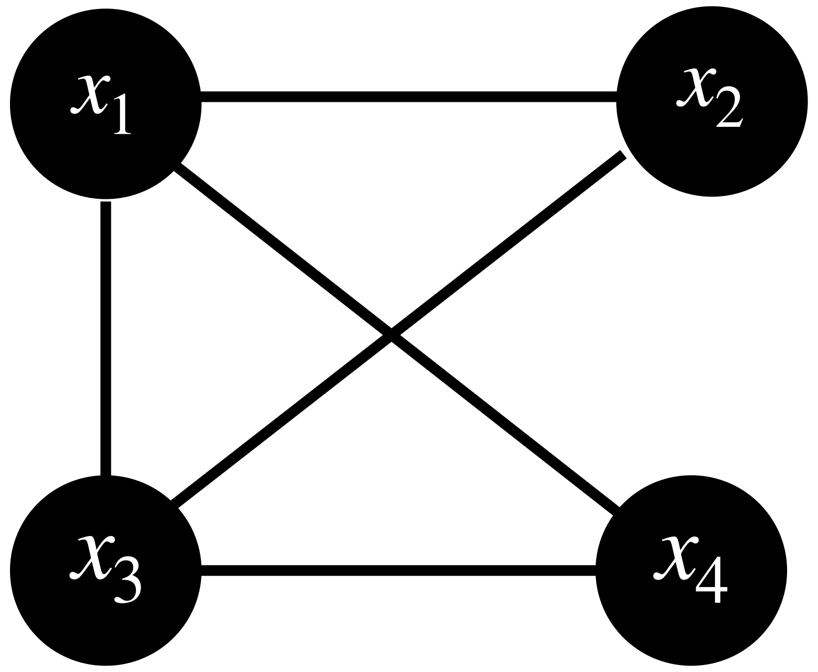
$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{H}^{(t-1)}\mathbf{W}_{self}^{(t)} + \mathbf{A}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\right),$$

...where the matrix $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ has the node representations at layer t.

MPNN layers apply a filter $\mathbf{Q} = \mathbf{I} + \mathbf{A}$, combined with some weight matrices and a non-linearity.

Convolution based on spectral properties of the graph, e.g., via the adjacency matrix! Other matrices?

Graph Laplacian



$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

$$\mathbf{D} \qquad \mathbf{A} \qquad \mathbf{L} = \mathbf{D} - \mathbf{A}$$

Property: Commutativity of the filter with the adjacency matrix $\mathbf{AQ} = \mathbf{QA}$ or Laplacian $\mathbf{LQ} = \mathbf{QL}$.

Symmetric Normalized Filters

Filters are typically normalized to ensure that they have bounded spectra, and thus ensure numerical stability.

Symmetric normalized Laplacian $$\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$$

Symmetric normalized adjacency matrix $$\mathbf{A}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

$$\mathbf{L}_{sym} = \mathbf{I} - \mathbf{A}_{sym}$$

Symmetric Normalized Filters

Filters are typically normalized to ensure that they have bounded spectra, and thus ensure numerical stability.

Symmetric normalized Laplacian $$\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$$

Symmetric normalized adjacency matrix $$\mathbf{A}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

$$\mathbf{L}_{sym} = \mathbf{I} - \mathbf{A}_{sym}$$

These matrices share the set $\mathbf{U}$ of eigenvectors and are symmetrically diagonalizable:

$$\mathbf{L}_{sym} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top \qquad \mathbf{A}_{sym} = \mathbf{U}(\mathbf{I} - \boldsymbol{\Lambda})\mathbf{U}^\top$$

...where $\boldsymbol{\Lambda}$ is the diagonal matrix containing the Laplacian eigenvalues.

Observation: Filters based on one of these matrices implies commutativity with the other.

Graph Convolutional Networks

Symmetric normalized adjacency matrix with self-loop (and variants) widely adopted as filters in practice:

$$\hat{\mathbf{A}} = \left(\mathbf{D} + \mathbf{I}\right)^{-\frac{1}{2}}\left(\mathbf{I} + \mathbf{A}\right)\left(\mathbf{D} + \mathbf{I}\right)^{-\frac{1}{2}}$$

This is the convolutional filter underlying the basic graph convolutional network (GCN) model:

$$\mathbf{H}^{(t)} = \sigma\left(\hat{\mathbf{A}}\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\right) \qquad \mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u) + N(v)}}\right)$$

Intuitively, in the base GCN model:

- $\hat{\mathbf{A}}$ enables messaging between neighbors and with node's self representation through the identity.
- $\hat{\mathbf{A}}$ is a well-defined convolution over graphs: commutativity with the adjacency matrix.
- Node's own embedding is treated identically to messages from other nodes: self-loops. Variations exist.

Graph Attention Networks

Learning Aggregation

Pre-defined, fixed aggregation schemes based on, e.g., graph structure:

$$\sum_{v \in N(u) \cup \{u\}} \mathbf{h}_v^{(t-1)} \qquad \sum_{v \in N(u)} \mathbf{W}\mathbf{h}_v^{(t-1)} \qquad \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u) + N(v)}}$$

Some learnable approaches to aggregation exist but uniform nevertheless.

Question: Can we learn to aggregate not necessarily uniformly across neighbors?

Idea: Use attention as a means to non-uniformly aggregate over the neighborhood.

Background: Attention models obtained strong results in, e.g., machine translation (Bahdanau et al., 2015).

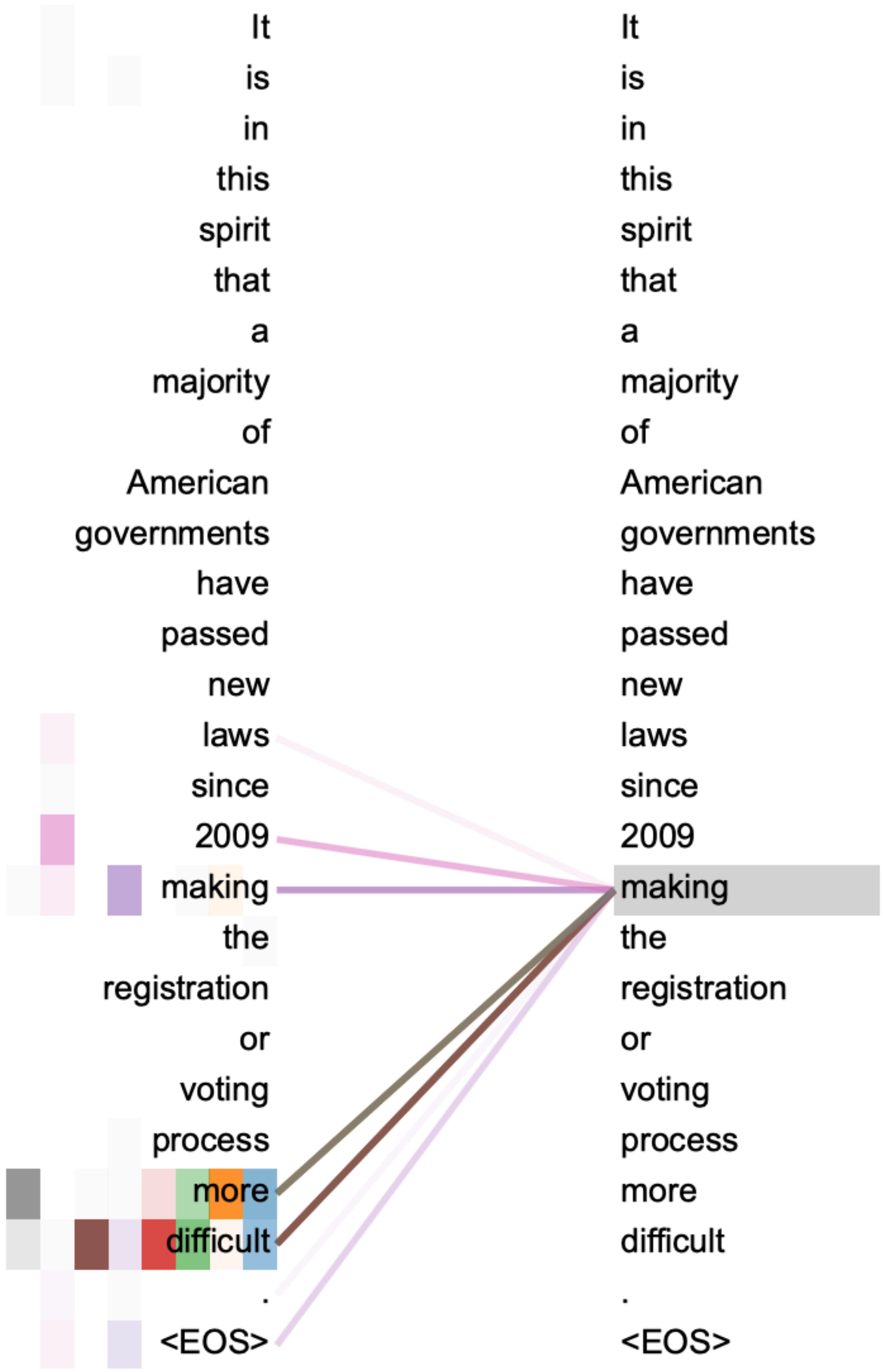
Attention

Attention: Allocate different weights to distinct inputs, based on their relevance to the learned task.

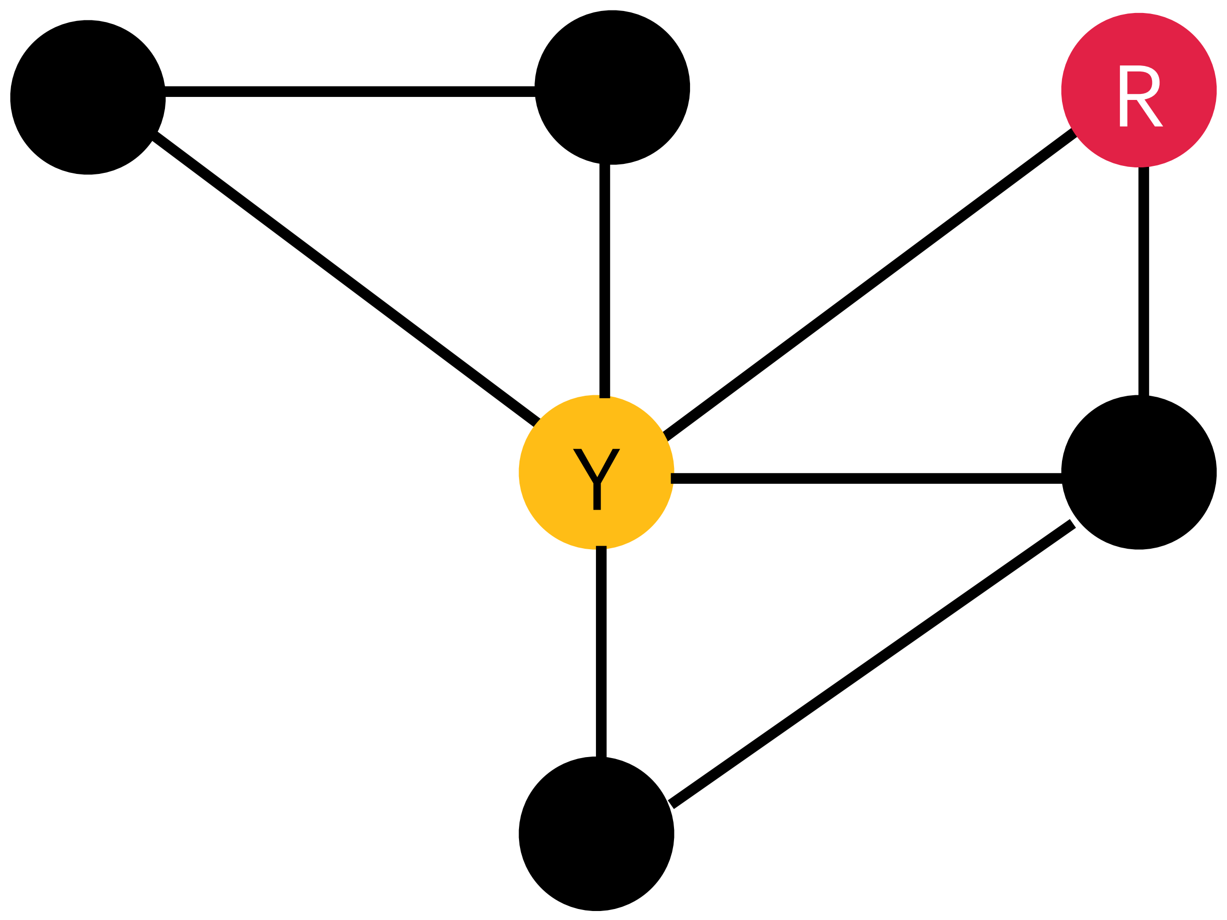
Transformer (Vaswani et al., 2017): Figure shows attention weights for the word 'making' encoding "making more difficult".

Breaking uniformity: Attend to more relevant tokens, rather than uniformly considering all possible tokens.

Graph attention: A node can benefit from weighing the relative importance of its neighbors.



Attention over Graphs



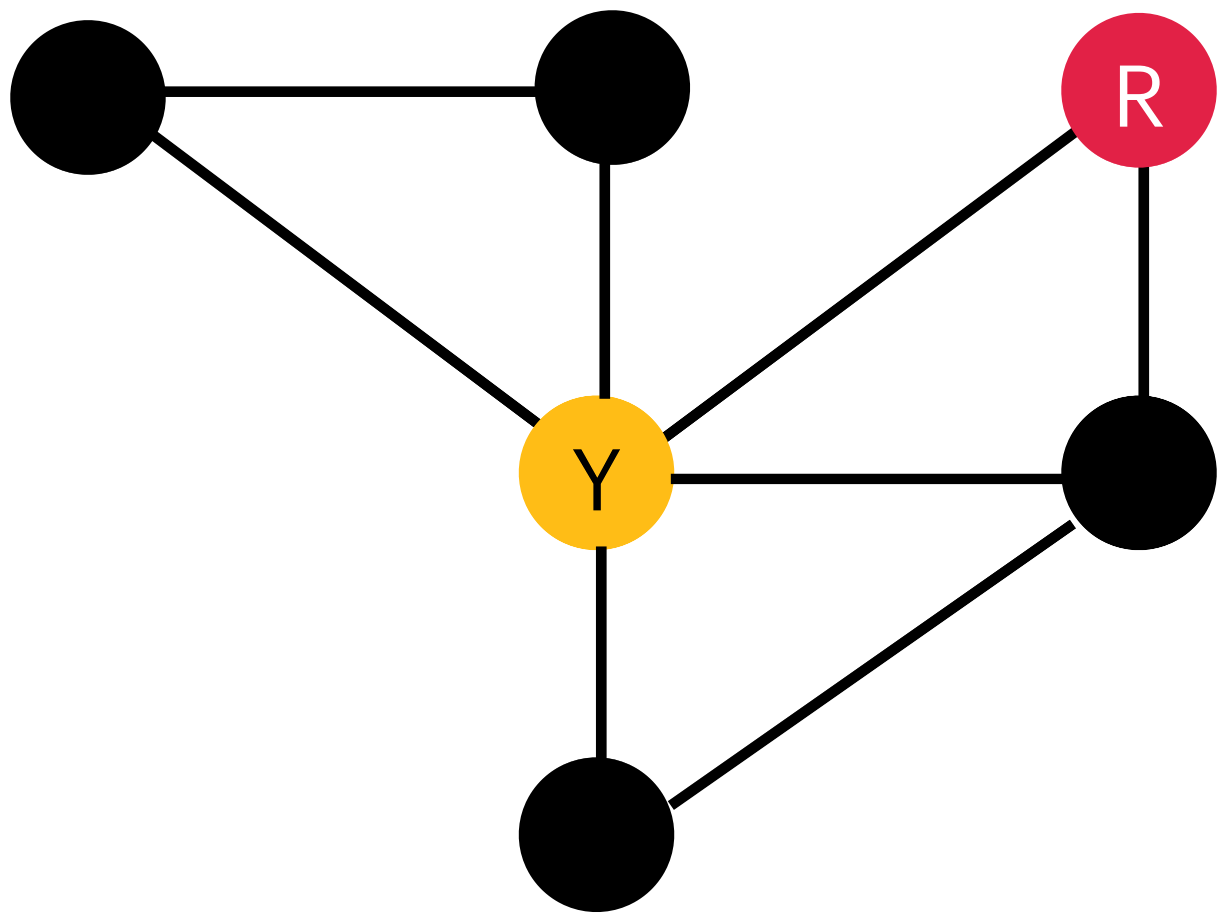
Example: Classify all nodes connected to a red node as true and every other node as false.

This task relies only to the fact that it is connected to a red node.

Neighborhood attention can produce a richer weighing of a node's neighbors, which results in potentially more descriptive and task-specific aggregation schemes.

Idea: Learn an attention weight for each neighbor: weighted aggregation functions.

Graph Attention Networks

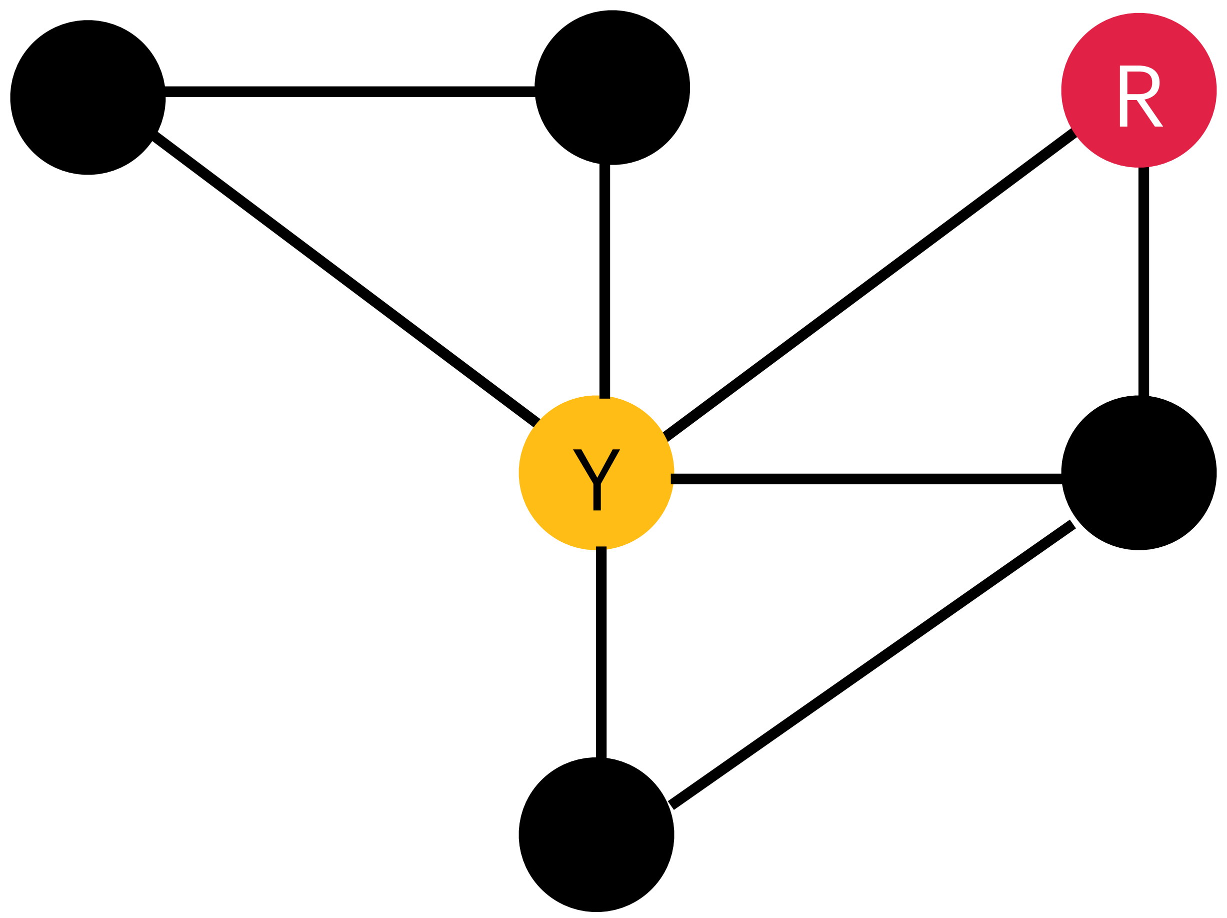


Graph attention networks (GAT) (Velickovic et al., 2018) apply weighted sum aggregation, and a pairwise node attention mechanism during message passing (using a self-loop approach):

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \alpha_{(u,v)}\, \mathbf{h}_v^{(t-1)}\right),$$

where $\alpha_{u,v}$ is the attention on a node $v \in N(u) \cup \{u\}$ when we aggregate information at node u.

What kind of Attention?

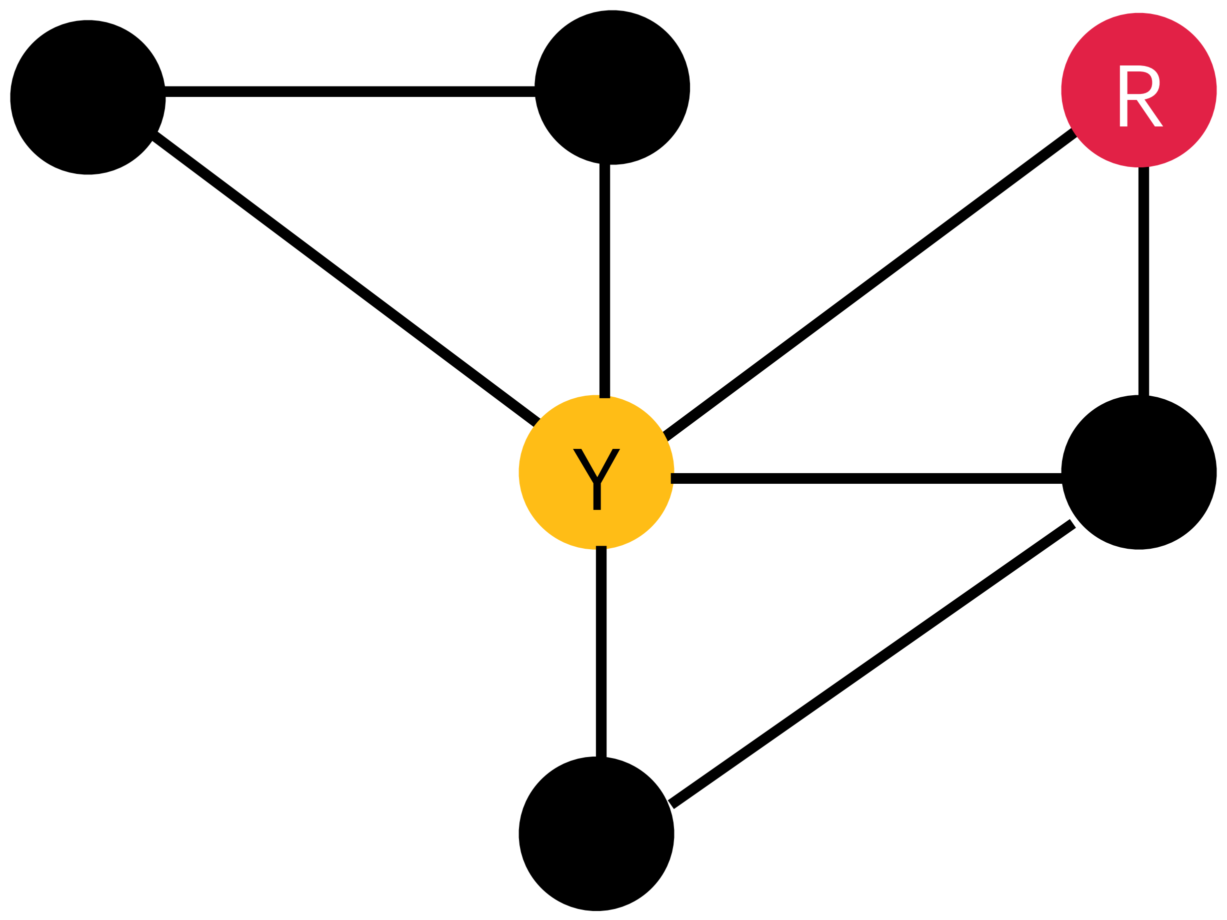


The attention weights $e_{u,v}$ between nodes u, v are normalized typically to yield final weights $\alpha_{u,v}$:

$$\alpha_{u,v} = \frac{\exp(e_{u,v})}{\sum_{v' \in N(u)} \exp(e_{u,v'})}$$

GAT: $e_{u,v} = \mathbf{a}^\top[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]$ Bilinear: $e_{u,v} = \mathbf{h}_u^\top \mathbf{W}\mathbf{h}_v$

Multi-Head Attention?



Multi-head attention: Learn multiple, distinct, independently parametrized attention weights.

Multi-head attention over graphs: Learn k attention weights $\alpha_{u,v,1}, \ldots, \alpha_{u,v,k}$ for the nodes u, v.

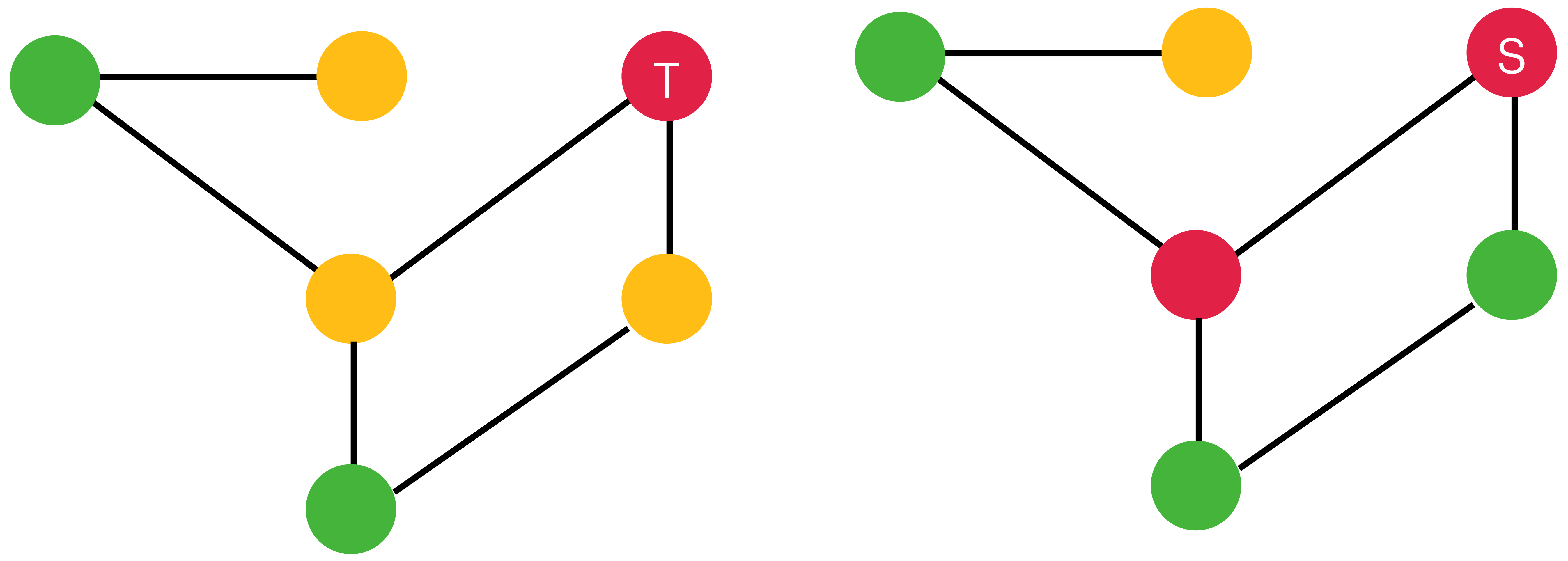
Node representations: This yields k node representations $\mathbf{h}_u[1], \ldots, \mathbf{h}_u[k]$ for each node u.

$$\mathbf{h}_u = \mathbf{h}_u[1] \oplus \ldots \oplus \mathbf{h}_u[k]$$

Transformer: Multiple attention heads to compute attention weights between all pairs of positions in the input. This coincides with GAT with multi-head attention on a fully connected graph as input.

Graph Isomorphism Networks

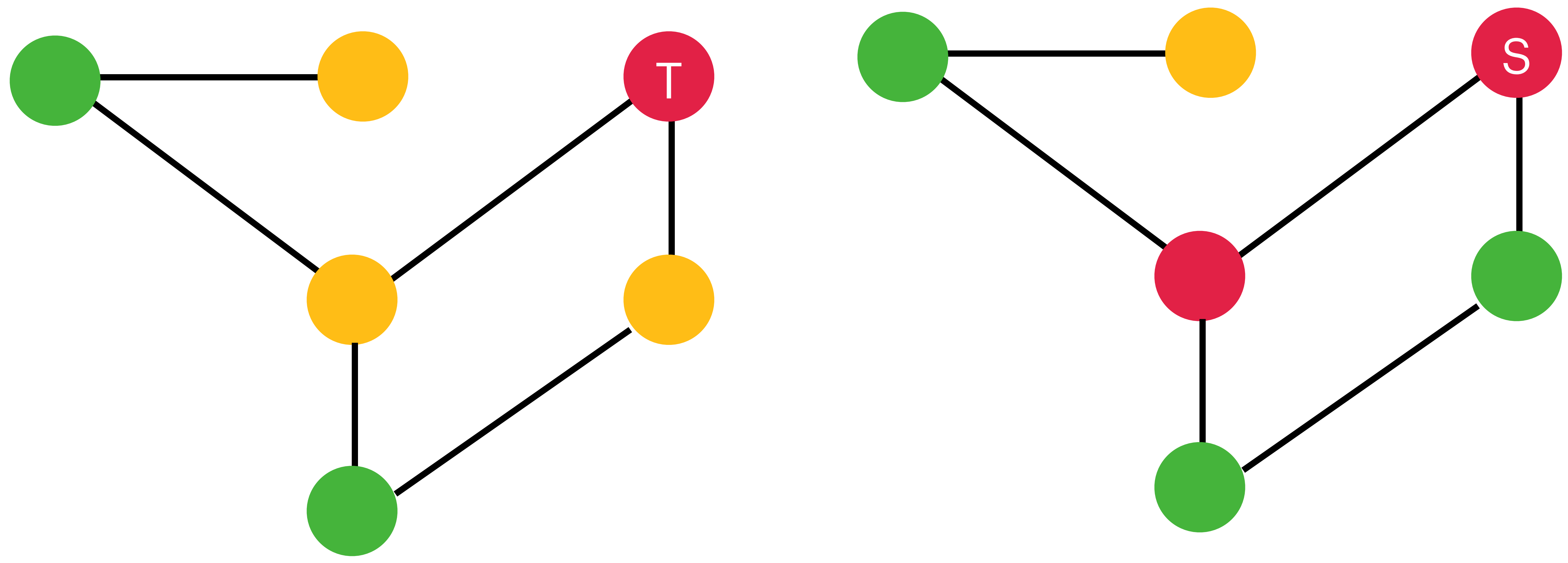
A Closer Look at Aggregation



Question: What is the impact of different choices of aggregation on the discrimination ability of GNNs?

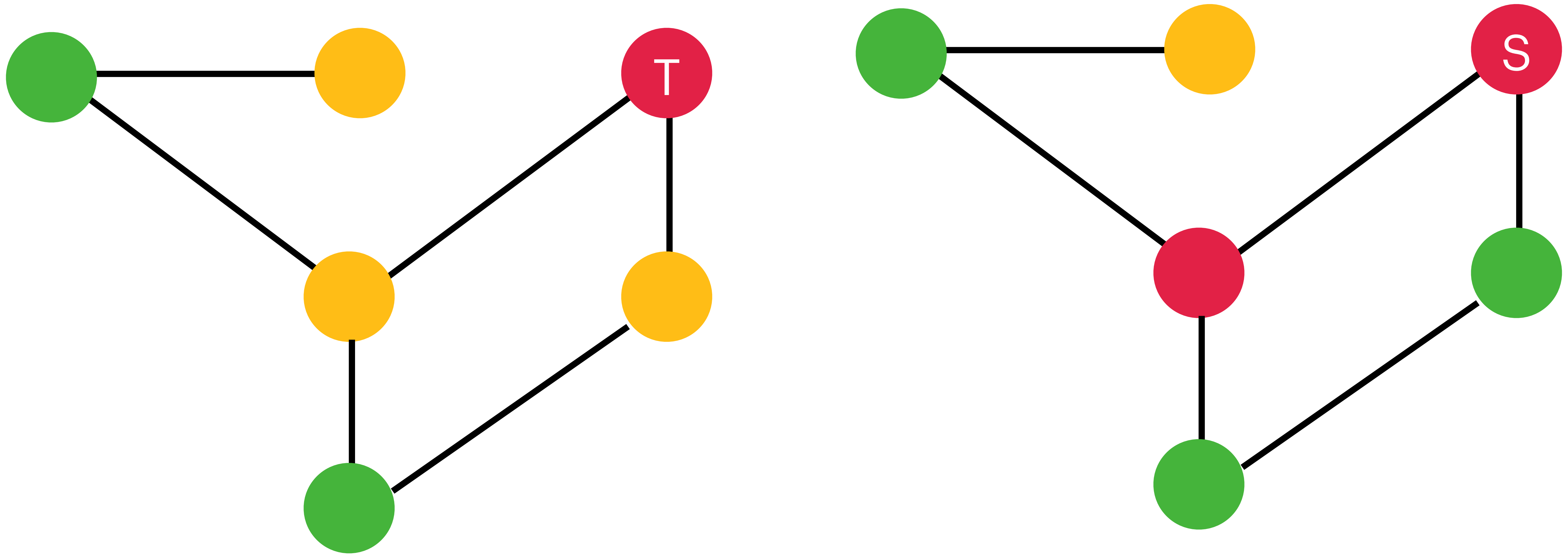
Task: Input graph with node types red, green and yellow, where the features are the RGB values. We consider a red node, and want to analyze how different functions aggregate neighbor messages.

A Closer Look at Aggregation



- **Sum**: Can discern between neighborhoods based on their sizes, but it can lead to false equality: In this example, sum cannot distinguish between a 2-yellow and a red-green neighborhood.
- **Mean**: Useful for bounding the range of aggregate messages, but cannot distinguish between neighbor sets such as 2-red and 3-red, as the mean operation eliminates cardinality.
- **Max**: Highlights a relevant element, but limited in discriminative ability. Considering red $<$ yellow $<$ green, then green is returned for any neighborhood involving at least 1 green node.

Aggregation and Expressiveness



Observation: An aggregation function must distinguish between distinct neighborhoods, and return different results given different neighborhood multisets.

Injective: The aggregation function must be injective relative to the neighborhood.

Expressive power: MPNNs are at their maximal expressiveness with injective functions (Xu et al.,2019).

Aggregation and Expressiveness

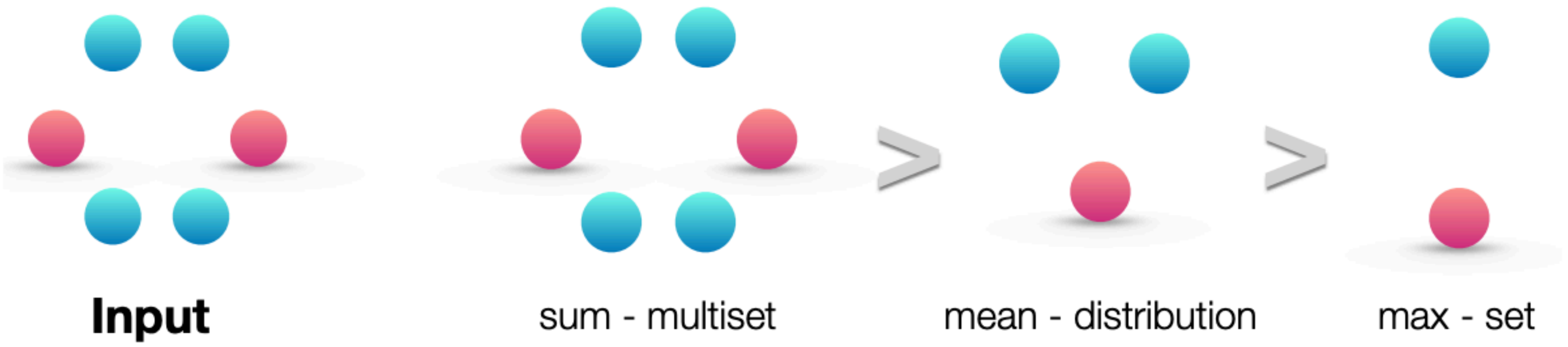
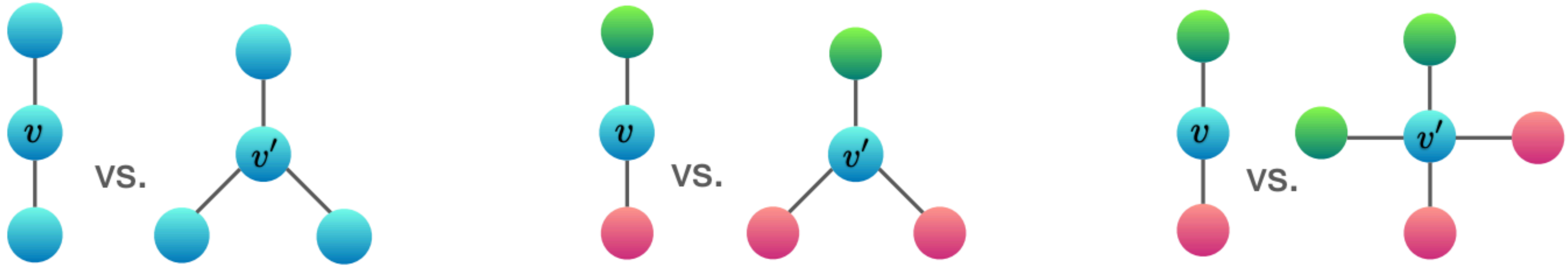


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset.** Left panel shows the input multiset, *i.e.*, the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities (reduces the multiset to a simple set).

(Xu et al., 2019)

Aggregation and Expressiveness

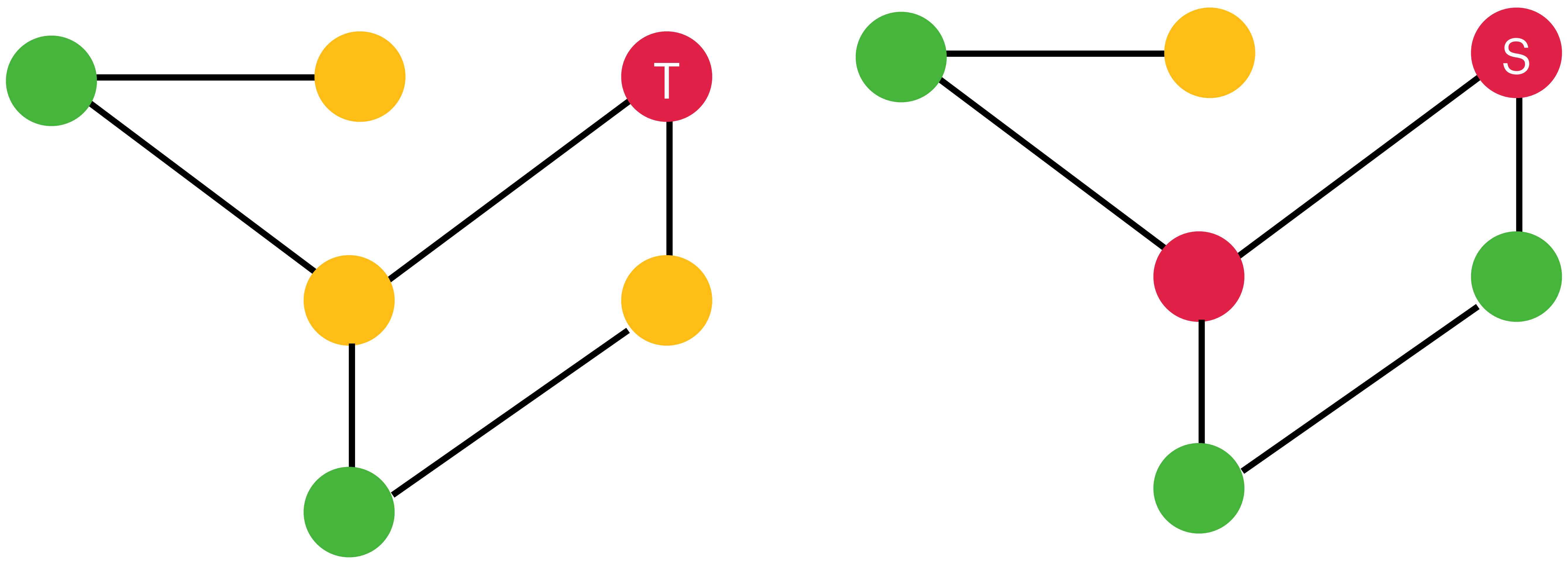


(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

Figure 3: **Examples of graph structures that mean and max aggregators fail to distinguish.** Between the two graphs, nodes v and v' get the same embedding even though their corresponding graph structures differ. Figure 2 gives reasoning about how different aggregators "compress" different multisets and thus fail to distinguish them.

(Xu et al., 2019)

Graph Isomorphism Networks

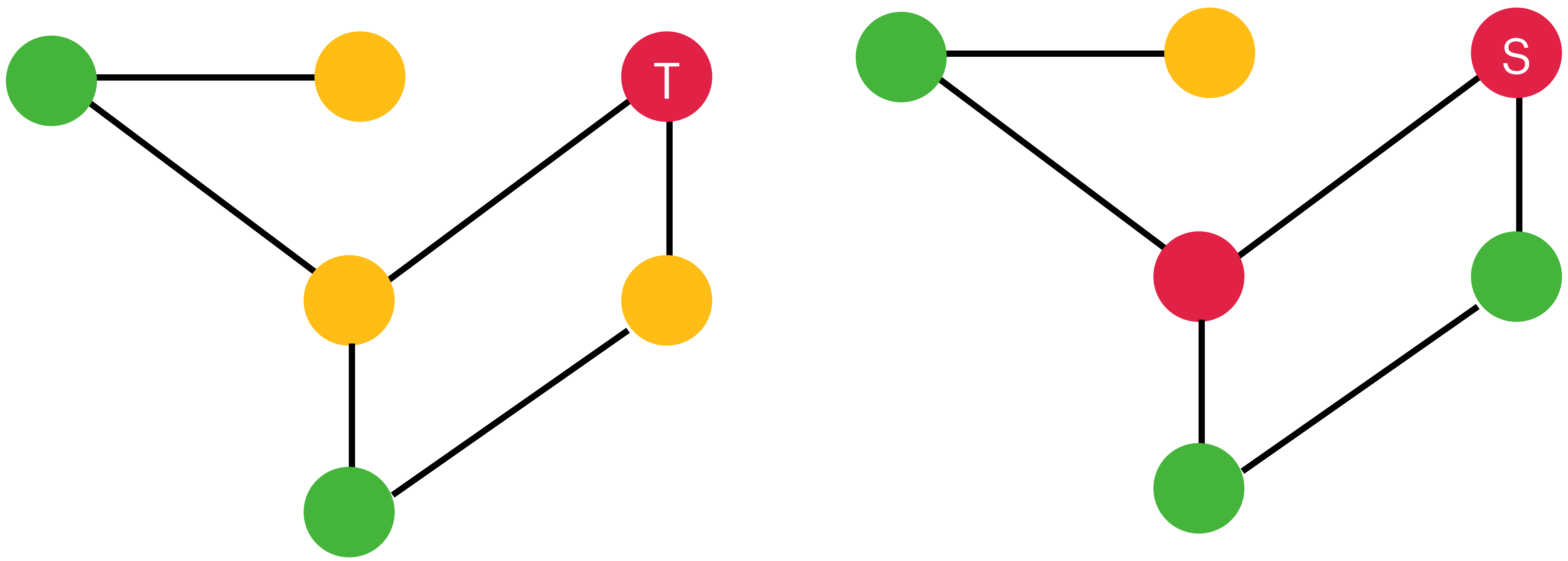


Idea: Let X be a bounded multi-set, ϕ and f some (expressive) non-linear functions, then the following

$$g = \psi\Big(\sum_{x \in X} f(x)\Big)$$

...defines an injective mapping.

Graph Isomorphism Networks

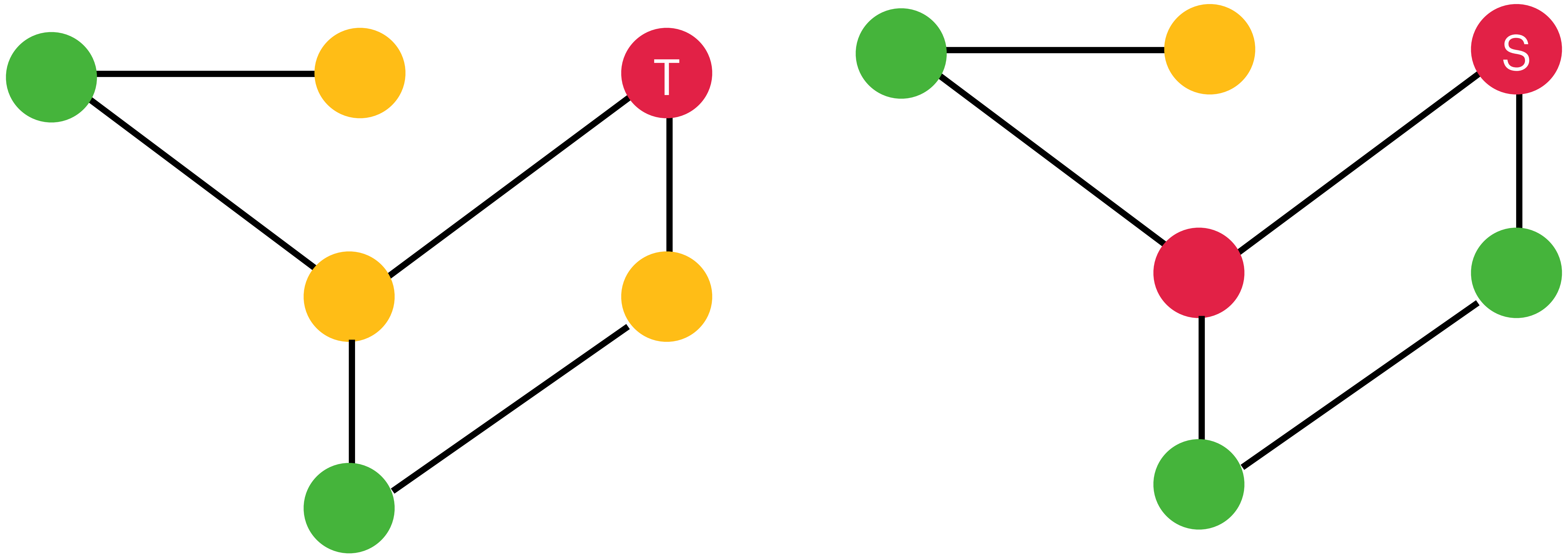


Example: Suppose we encode nodes states as $(R, G, Y)^T$

- $f(R) = (1,0,0)^T$, $f(G) = (0,1,0)^T$, $f(Y) = (0,0,1)^T$
- $g(\{\{Y, Y\}\}) = (0,0,2)^T$ and $g(\{\{R, G\}\}) = (1,1,0)^T$

$$g = \psi\Big(\sum_{x \in X} f(x)\Big)$$

Graph Isomorphism Networks

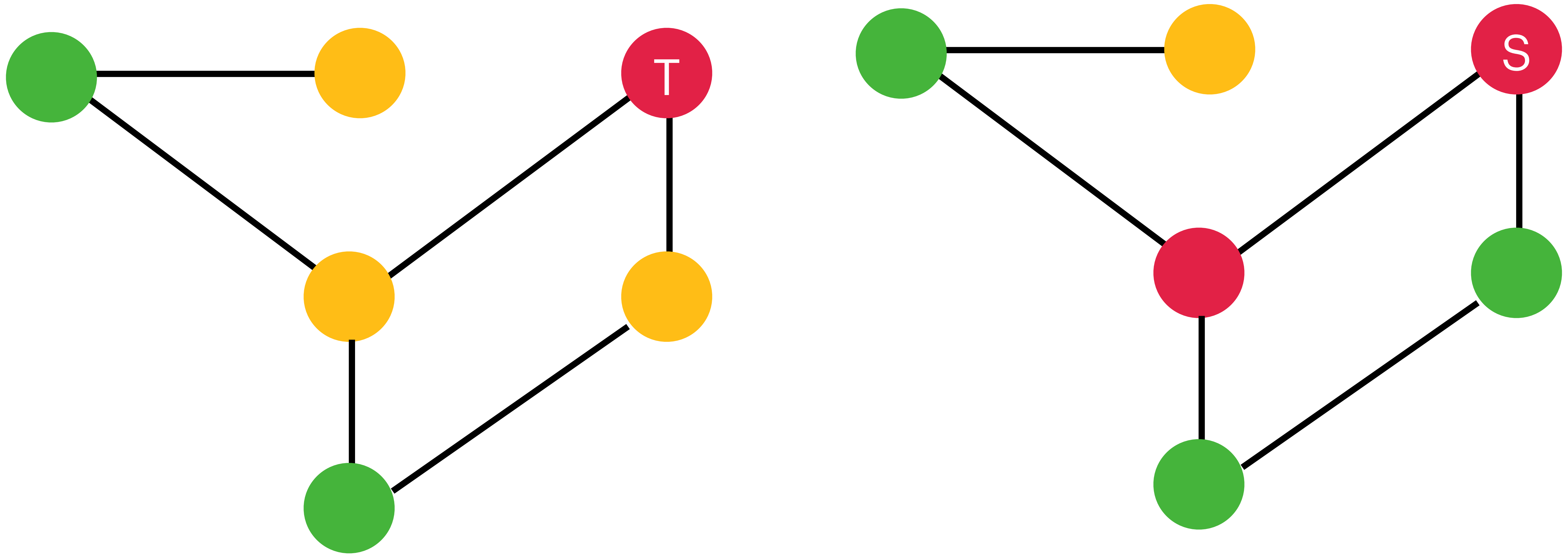


[**Lemma 5 & Corollary 6,** (Xu et al., 2019)] For a countable set $\mathscr{X}$, there exists a function $f : \mathscr{X} \to \mathbb{R}^n$ such that for any choice of ϵ, the function

$$g(c, X) = \psi\Big((1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)\Big)$$

is unique for each pair (c, X), where $X \subset \mathscr{X}$ is a multiset of bounded size and $c \in X$.

Graph Isomorphism Networks

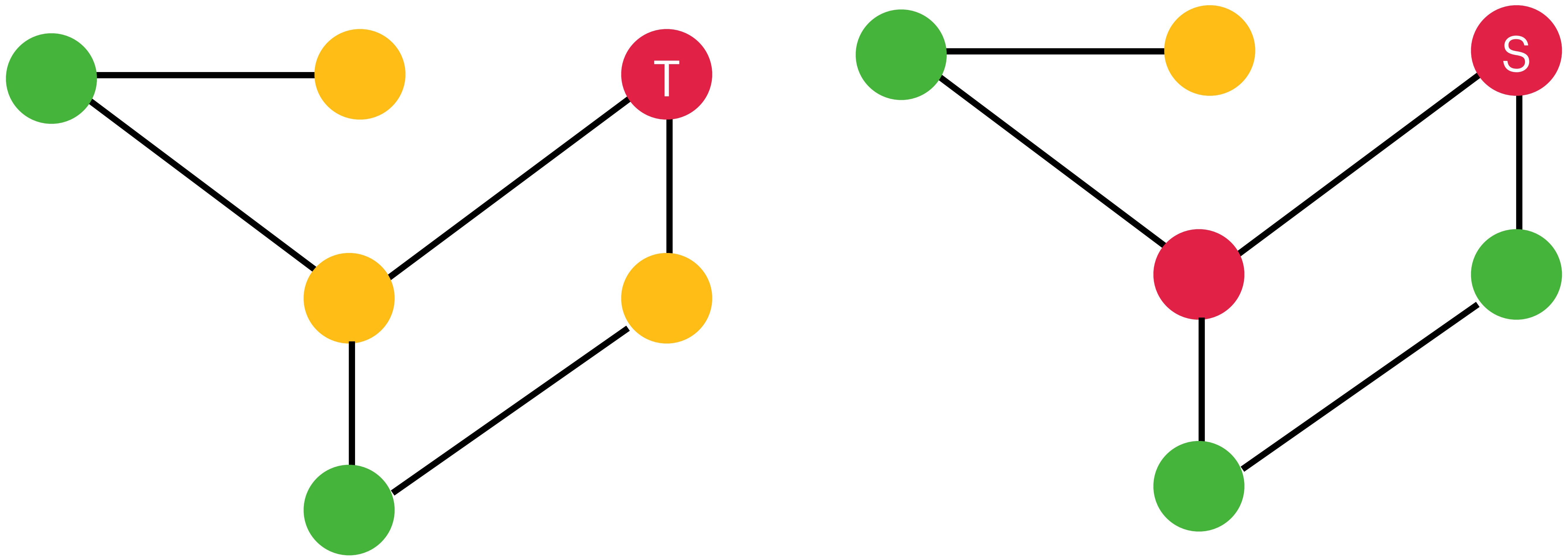


We can use MLPs to learn these functions, as MLPs are universal approximators (Hornik et al., 1989):

$$\mathbf{h}_u^{(t)} = MLP_\psi\Big((1+\epsilon)\cdot MLP_f\big(\mathbf{h}_u^{(t-1)}\big), \sum_{v\in N(u)} MLP_f\big(\mathbf{h}_v^{(t-1)}\big)\Big)$$

...which yields another instance of MPNNs.

Graph Isomorphism Networks



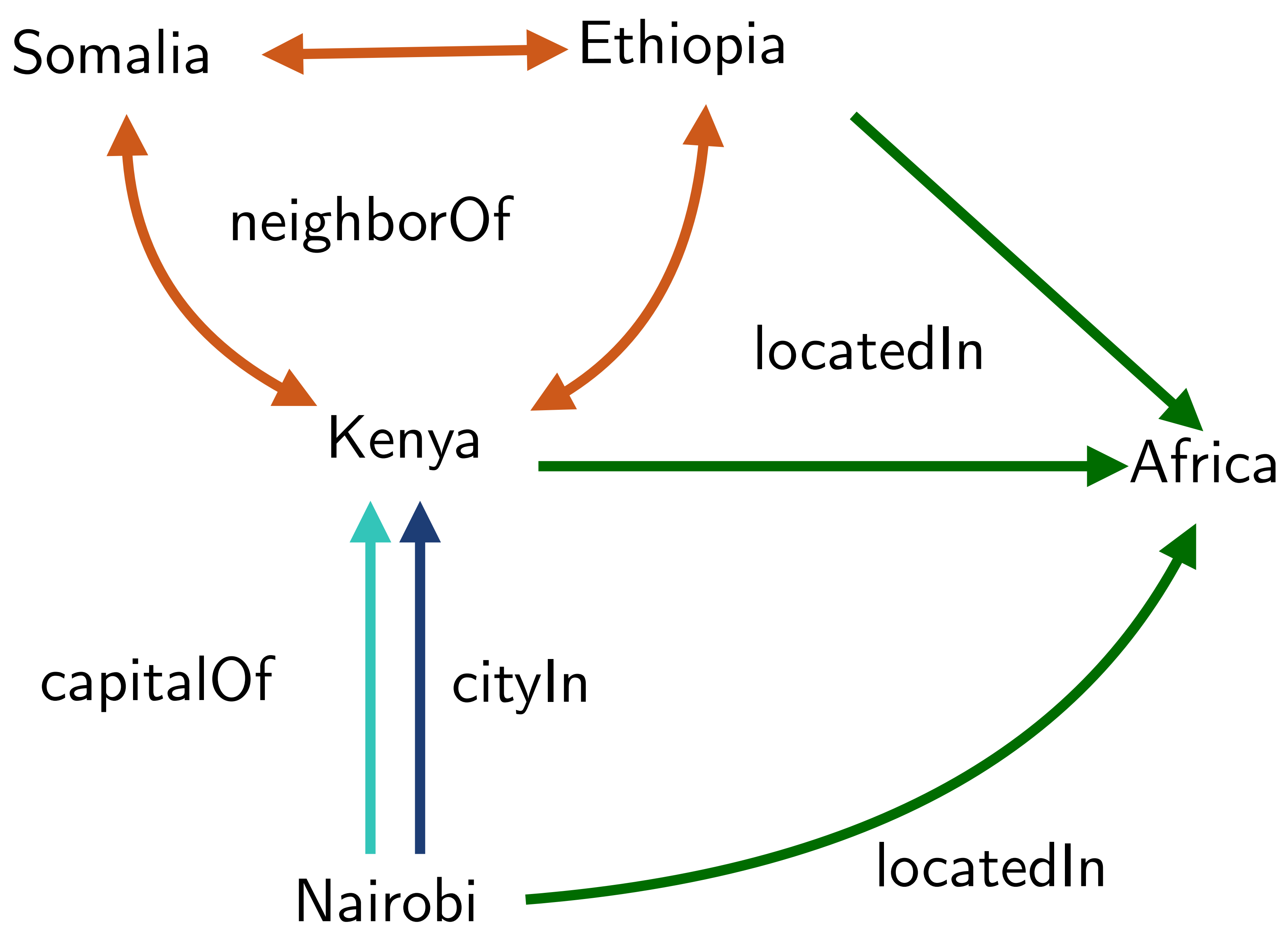
Graph isomorphism networks (GINs) update the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively as:

$$\mathbf{h}_u^{(t)} = MLP\left((1+\epsilon) \cdot \mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)} \right)$$

...by setting $MLP = f^{(t+1)} \circ \psi^{(t)}$ and assuming the features are encoded as one-hot initially.

Relational Message Passing

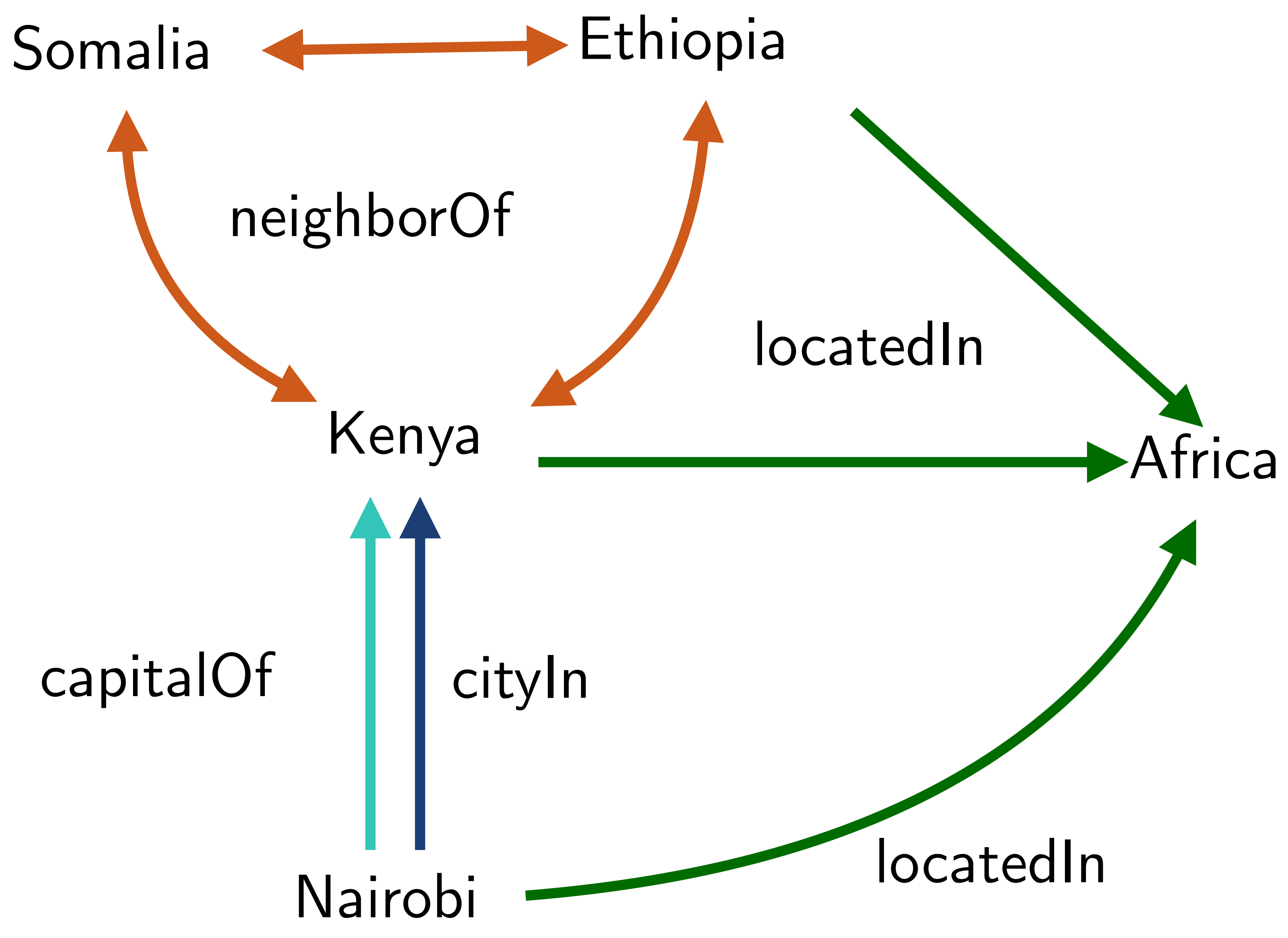
Relational Graphs



Relational graphs: Relevant for a variety of tasks, e.g., entity/node classification, KG completion.

GNNs are extended to the multi-relational setting to deal with multi-relational graphs.

Relational Graphs

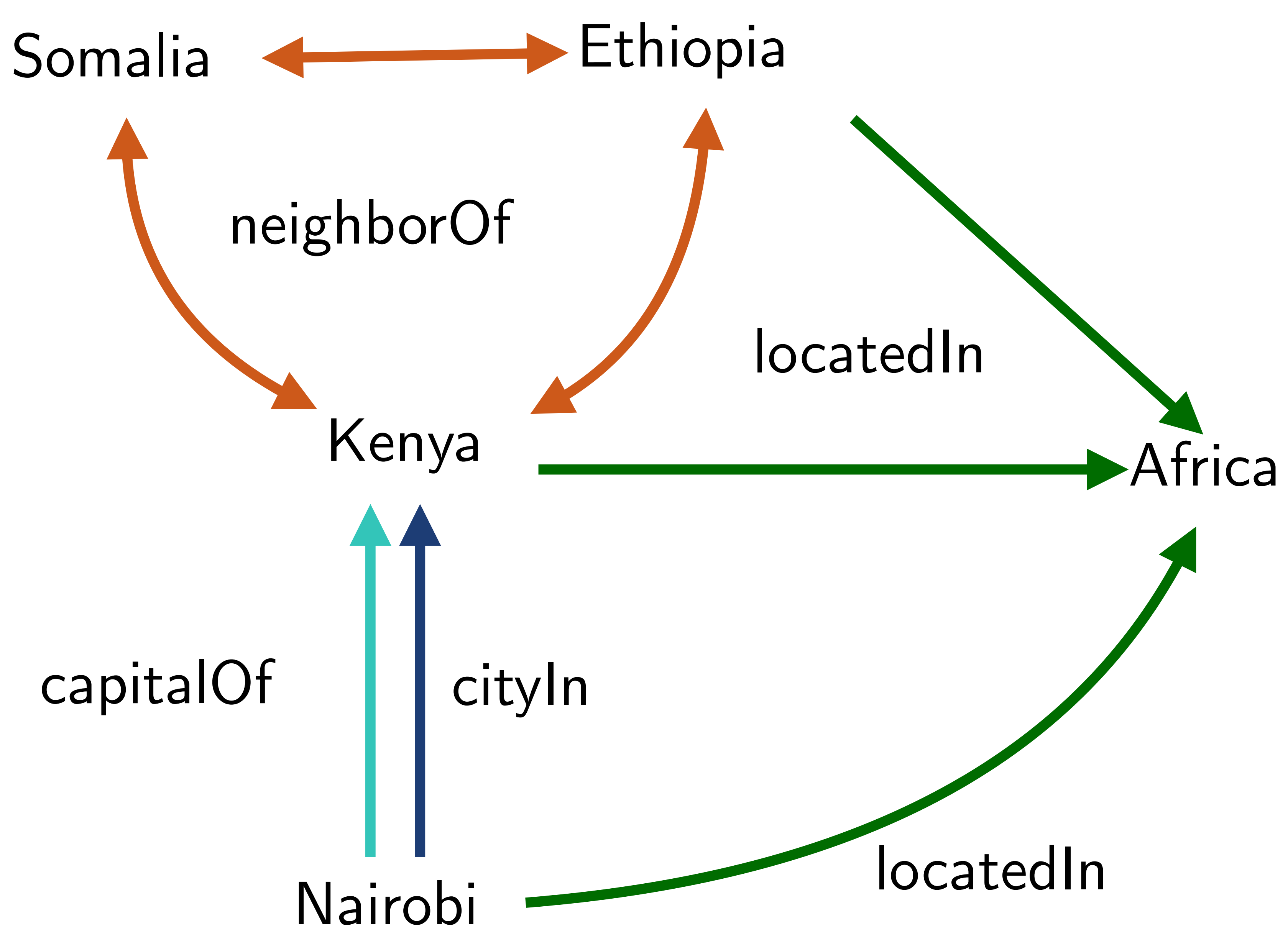


The model rGCNs (Schlichtkrull et al., 2018) defines a relation-specific message passing:

$$\mathbf{h}_u^{(t)} = \sigma\Big(\sum_{r \in \mathbf{R}} \sum_{v \in N^r(u)} \big(\frac{1}{c_{u,r}}\big) \mathbf{W}_r^{(t)} \mathbf{h}_v^{(t-1)} + \mathbf{W}_{self}^{(t)} \mathbf{h}_u^{(t-1)}\Big)$$

where $r \in \mathbf{R}$ is a relation, and $c_{u,r}$ is a normalization constant.

Relational Graphs



The rGCN model applies to both for node/graph classification but also KG completion.

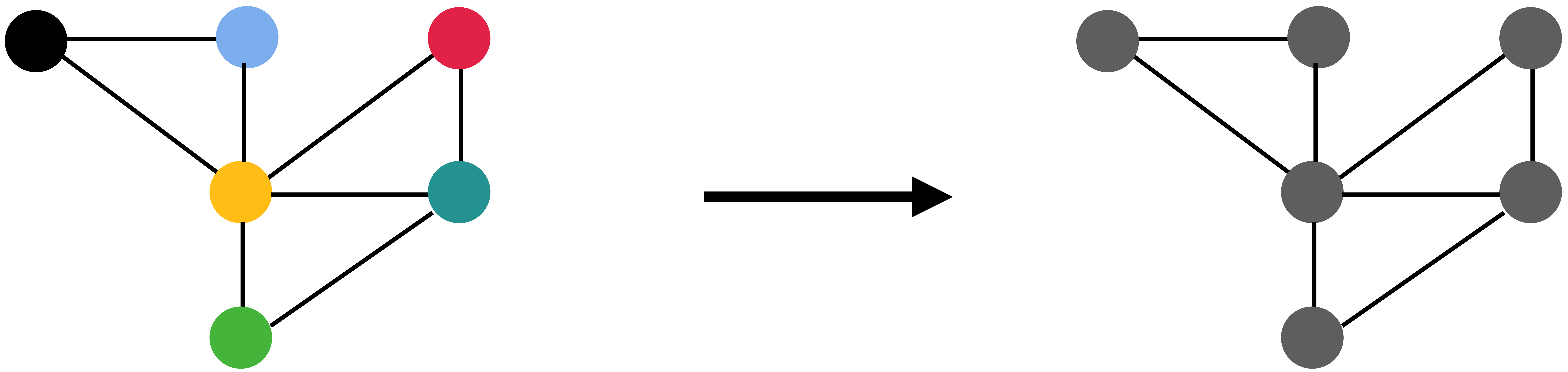
The learned embeddings are used as the entity embeddings and fed to a decoder, e.g., DistMult.

Note that rGCNs combine many aspects of this course: shallow KGC models and GNNs!

rGCN performs usually worse than shallow tools which motivated a line of work, e.g., GrAIL...

Limitations of Message Passing Neural Networks

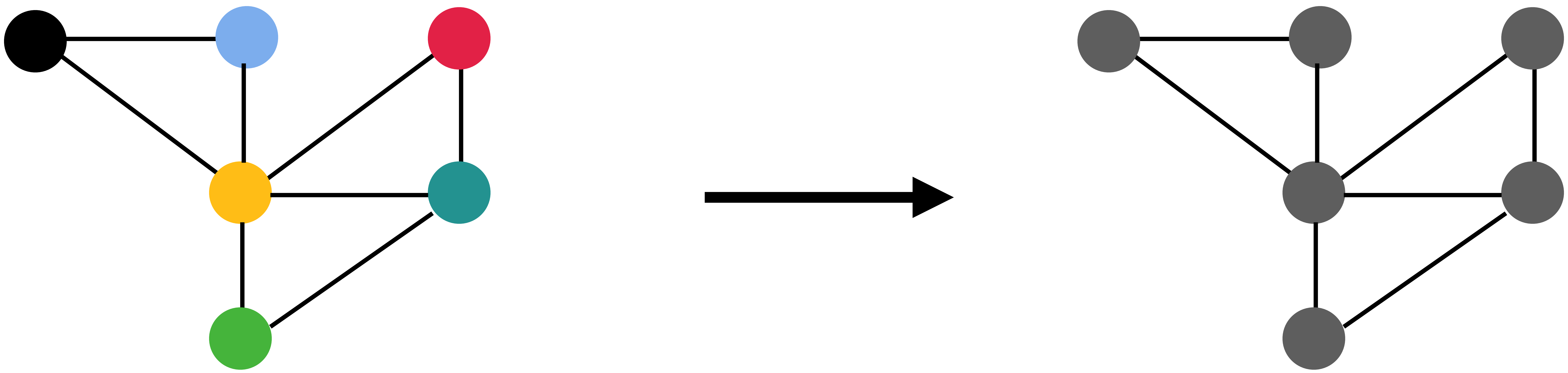
Over-smoothing



Over-smoothing: The representations of the nodes in the graph become indistinguishable after several message passing iterations (Li et al., 2018).

Long-range dependencies: Hard to make meaningful predictions — especially for deep GNN models, where the goal is to pass information across many layers so as to capture long-range dependencies.

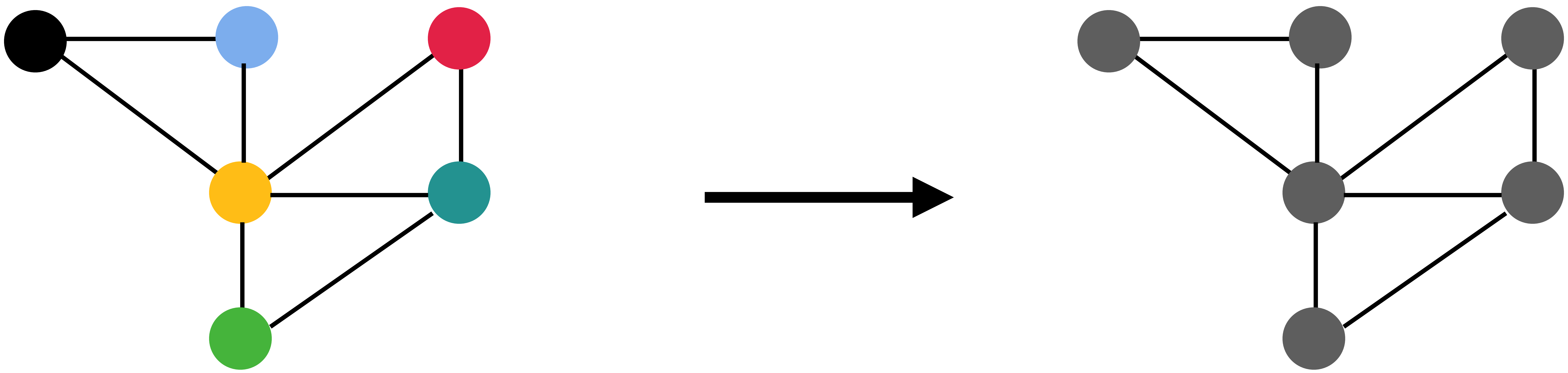
Over-smoothing



Intuition: Messages aggregated from the neighbors become too prominent, rendering the effect of the embeddings from the previous layers less and less important.

Practice: Significant performance degradation has been observed when stacking many layers on GNNs (Kipf & Welling, 2017); especially for GCNs (Li et al., 2018). Models such as GGNNs are somewhat better...

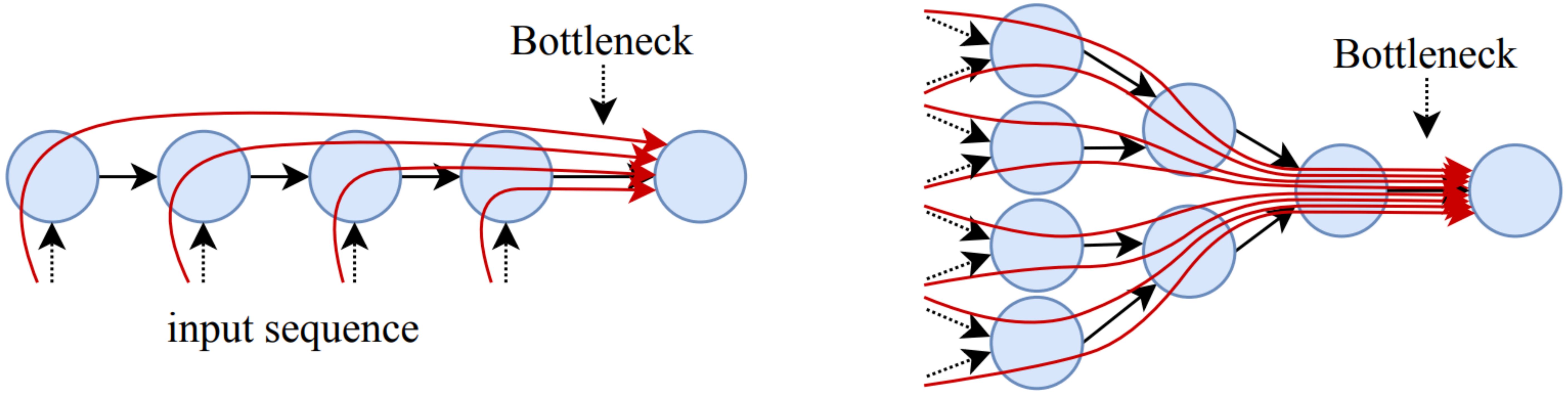
Over-smoothing



(**Theorem 3**, (Xu et al., 2018)) Informally, with a k-layer GCN, the influence of a node u on node v is proportional the probability of reaching node v on a k-step random walk starting from node u.

To partially alleviate over-smoothing: Concatenate each node's previous representation with the output of the combine function to preserve information from previous rounds.

Over-squashing



(a) The bottleneck of RNN seq2seq models

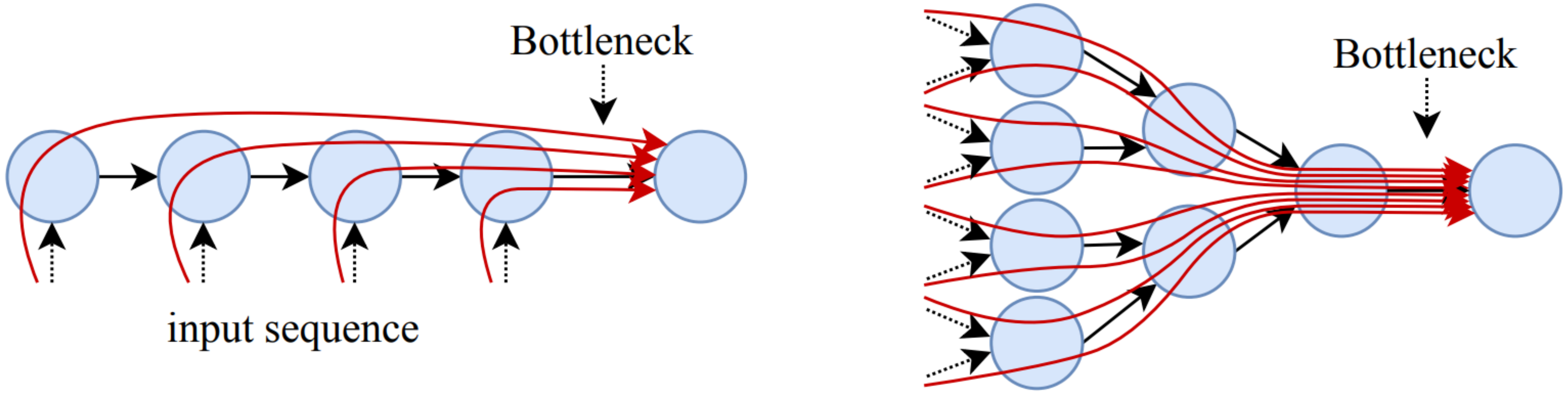
(b) The bottleneck of graph neural networks

Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Over-squashing (Alon and Yahav, 2021): The number of nodes in each node's receptive field grows exponentially, which is eventually compressed into fixed-length node state vectors, hence over-squashing information.

Long-range: Failure in propagating messages flowing from distant nodes - learning only from short-range signals.

Over-squashing



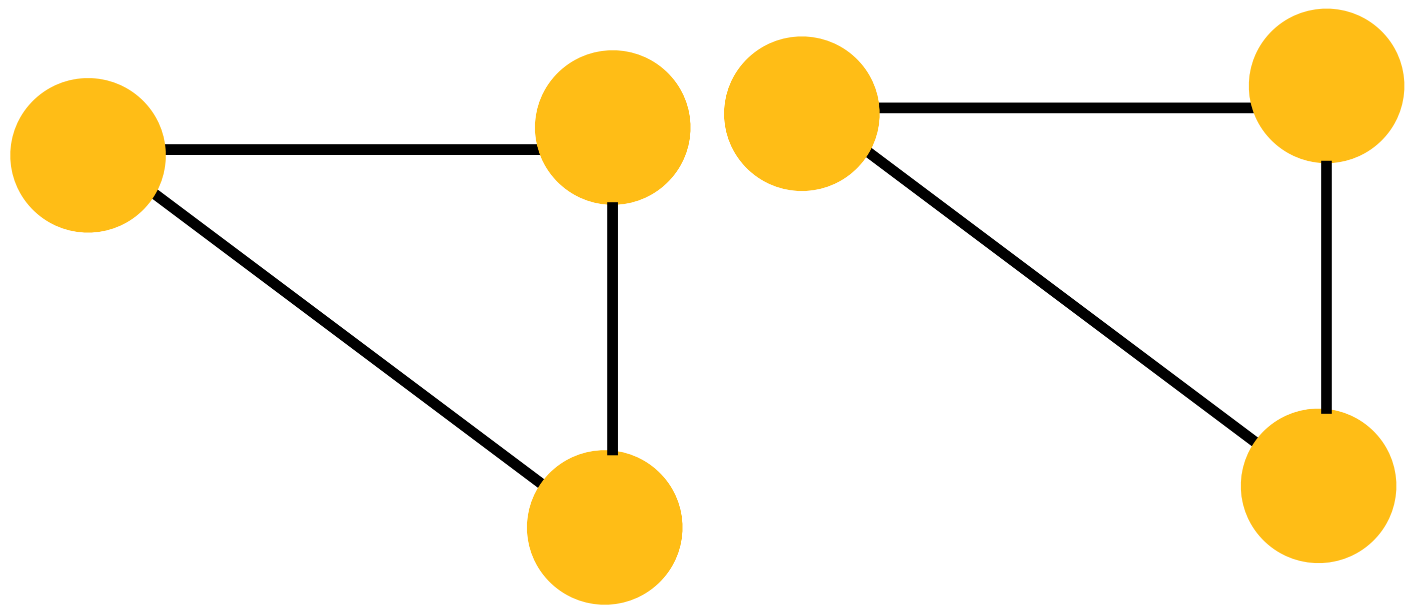
(a) The bottleneck of RNN seq2seq models (b) The bottleneck of graph neural networks

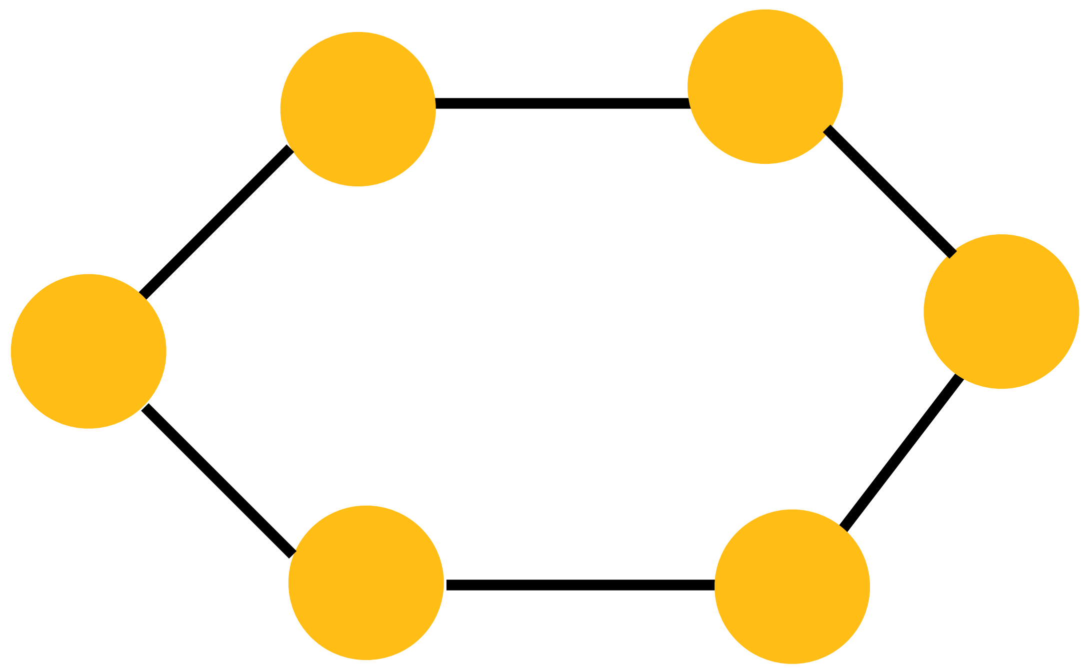
Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Practice: Poor performance when the task depends on long-range interactions, e.g., reachability task on graphs require as many iterations as the diameter of the graph, as otherwise it will suffer from under-reaching.

Global Information: Global feature computation can alleviate the issue to some extent. Alon and Yahav (2021) report improvements by using an additional fully connected layer.

Expressive Power





Expressive power: MPNNs is limited by the 1-WL graph isomorphism test

Example: Any MPNN learns the same embeddings for the graphs shown

This is the topic of the next lecture.

Summary

- An historical overview of graph neural networks:
  - **Gated graph neural networks**: graphs as sequences — gated units as the combine function.
  - **Graph convolutional networks**: each iteration of message passing is a convolution.
  - **Graph attention networks**: distinguish messages from neighbors via attention
  - **Graph isomorphism network**: injective aggregation
- Each of these models fall into the MPNN framework of Gilmer et al, (2017).
- Additional reading material: This lecture is partially based on Chapters 5 - 7 of Hamilton, (2020).
- We have not identified the expressive power of MPNNs: Lecture 5.
- There are a plethora of other GNN models, beyond MPNNs: Lecture 6.

References

- T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
- M. D. Zeiler, R. Fergus. Visualizing and Understanding Convolutional Networks, *ECCV*, 2014.
- Y. Li, D. Tarlow, M. Brockschmidt, and R.S. Zemel. Gated graph sequence neural networks. *ICLR*, 2016.
- P. Velickovic, G. Cucurull, A. Casanova, A. Romero,P. Lio, and Y. Bengio. Graph attention networks. *ICLR* 2018.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS*, 2017.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.
- W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NIPS*, 2017.
- J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *ICML*, 2017.
- M.Defferrard, X. Bresson, P. Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *NIPS*, 2016.
- J. Bruna, W. Zaremba, A. Szlam, Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *ICLR*, 2014.

References

- K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *IJCNLP*, 2015.
- H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. *ICML*, 2016.
- A. Santoro, D. Raposo, D.G.T.Barrett, M. Malinowski, R. Pascanu, P.W. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *NIPS*, 2017.
- R.L. Murphy, B. Srinivasan, V.A. Rao, and B. Ribeiro. Relational Pooling for Graph Representations. *ICML*, 2019.
- H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *NeurIPS*, 2019.
- C. Morris, M. Ritzert, M. Fey, W. Hamilton,J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. *AAAI*, 2019.
- W. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 2017.
- R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. *SDM*, 2021.
- R. Abboud, İ. İ. Ceylan, M. Grohe, T. Lukasiewicz, The Surprising Power of Graph Neural Networks with Random Node Initialization, *IJCAI*, 2021

References

- M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. *IJCNN*, 2005.
- F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks* 20(1):61–80, 2009.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Uri Alon, Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications, *ICLR*, 2021.
- Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, 2018.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.
- M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van den Berg, I. Titov, M. Welling, Modeling Relational Data with Graph Convolutional Networks, *ESWC*, 2018.
- K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In ICML, 2018.