DSC250: Advanced Data Mining

Graph Neural Networks

Zhiting Hu

Lecture 13, Feb 18, 2025

UC San Diego

HALICIOĞLU DATA SCIENCE INSTITUTE

Outline

- Graph neural networks

- Presentation
  - Yuan Lu, Songyao Jin: "Auto-Encoding Variational Bayes”
  - Shweta Nalluri, Keertana Kappuram: “Multi-task retriever fine-tuning for domain-specific and efficient RAG”
  - Jingman Wang, Jiayue Xu: “LLM-Enhanced Data Management”
  - Shanglin Zeng, Tianle Wang: “Learning Concise and Descriptive Attributes for Visual Recognition”

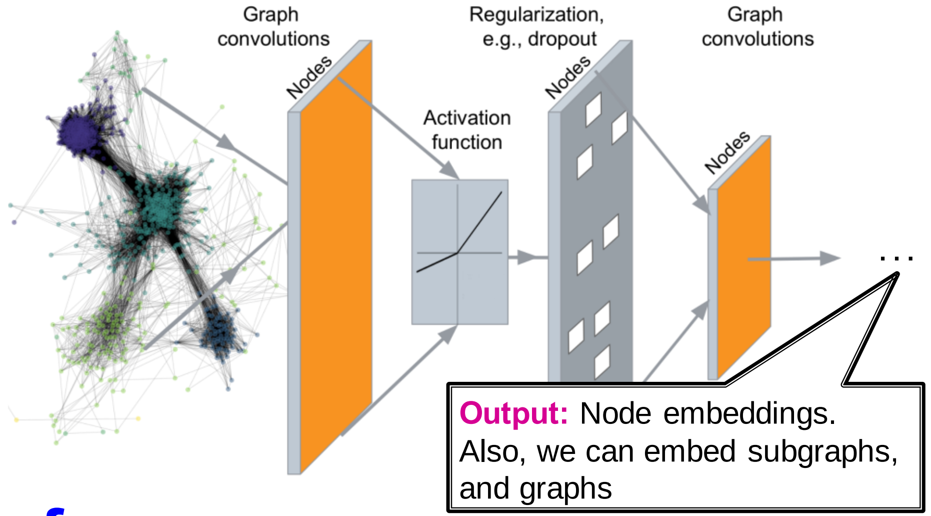
Recap: Summary

- **Encoder + Decoder Framework**
  - Shallow encoder: embedding lookup
  - Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
  - We will cover deep encoders in the GNNs

  - **Decoder:** based on node similarity.
  - **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for node pairs (u, v) that are **similar**

Recap: Deep Graph Encoders

- Encoding based on graph neural networks



Output: Node embeddings. Also, we can embed subgraphs, and graphs

$$\mathrm{ENC}(v) =$$ **multiple layers of non-linear transformations based on graph structure**

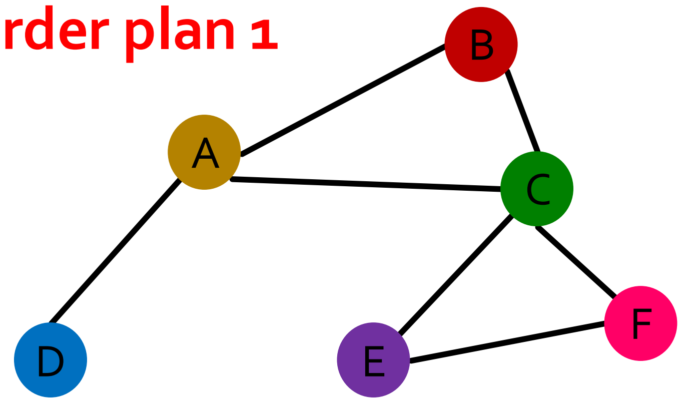
v.s. **Shallow Encoder:**

$$\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

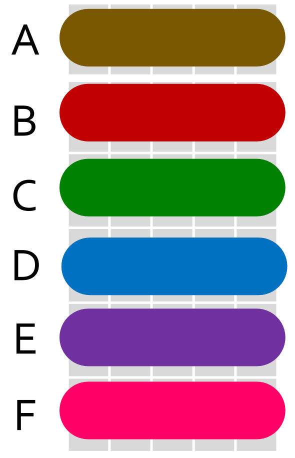
Recap: Permutation Invariance

- **Graph does not have a canonical order of the nodes!**

Order plan 1



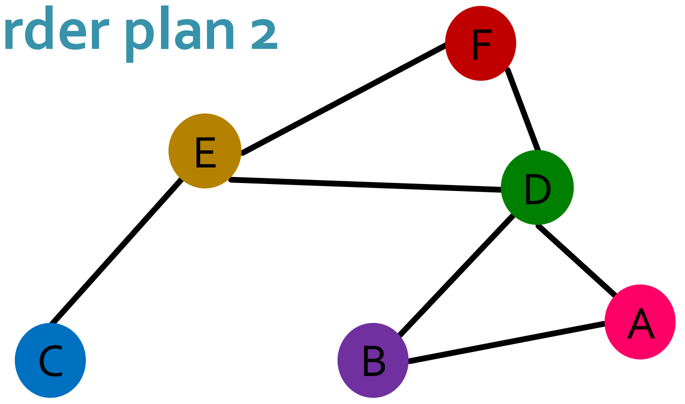
Node features X_1



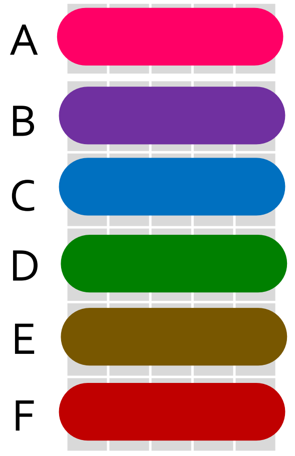
Adjacency matrix A_1

	A	B	C	D	E	F
A	⬛	🟦	🟦	🟦	⬛	⬛
B	🟦	⬛	🟦	⬛	⬛	⬛
C	🟦	🟦	⬛	⬛	🟦	🟦
D	🟦	⬛	⬛	⬛	⬛	⬛
E	⬛	⬛	🟦	⬛	⬛	🟦
F	⬛	⬛	🟦	⬛	🟦	⬛

Order plan 2



Node features X_2

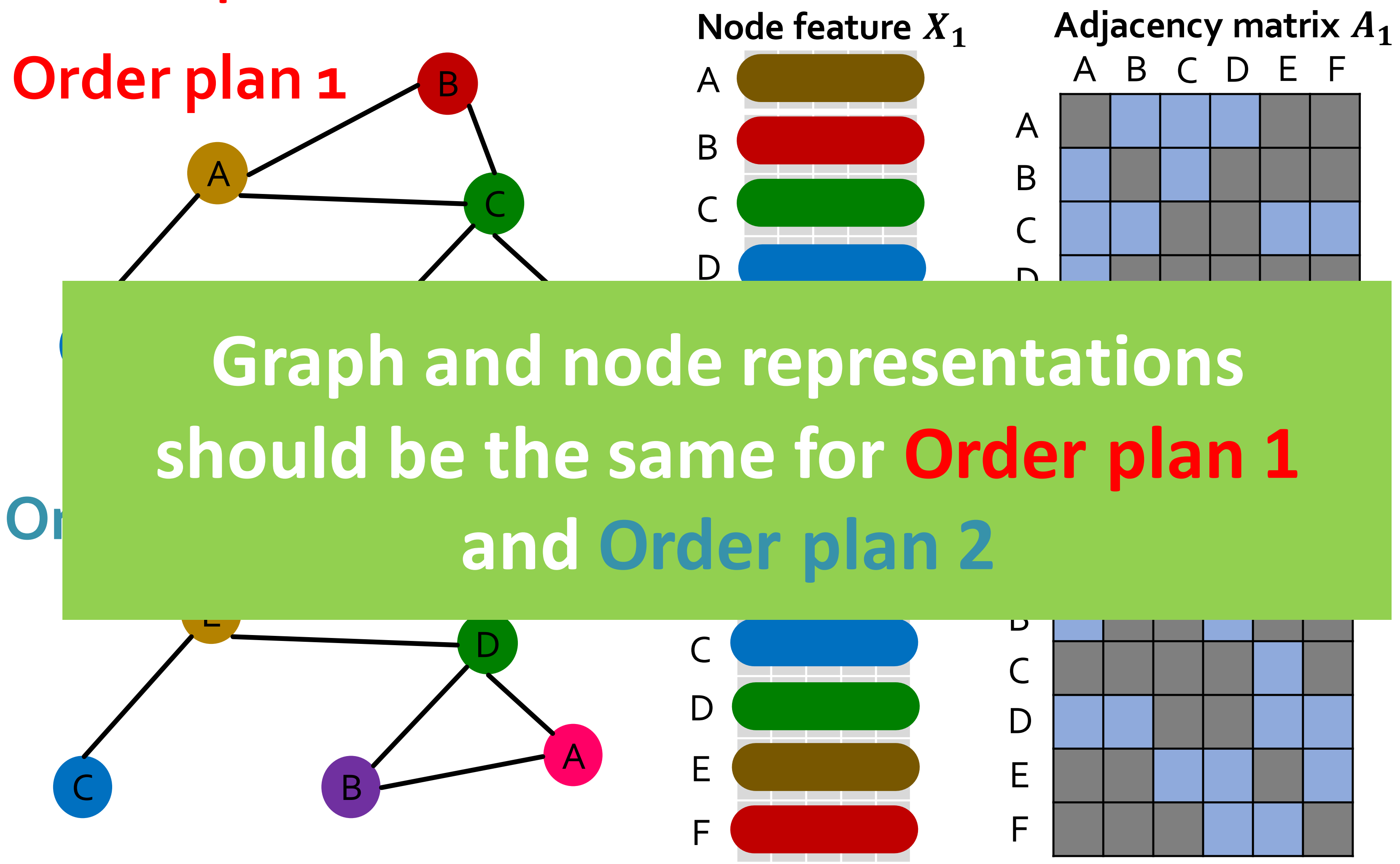


Adjacency matrix A_2

	A	B	C	D	E	F
A	⬛	🟦	⬛	🟦	⬛	⬛
B	🟦	⬛	⬛	🟦	⬛	⬛
C	⬛	⬛	⬛	⬛	🟦	⬛
D	🟦	🟦	⬛	⬛	🟦	🟦
E	⬛	⬛	🟦	🟦	⬛	🟦
F	⬛	⬛	⬛	🟦	🟦	⬛

Recap: Permutation Invariance

- **Graph does not have a canonical order of the nodes!**



Graph and node representations should be the same for Order plan 1 and Order plan 2

Recap: Permutation Invariance

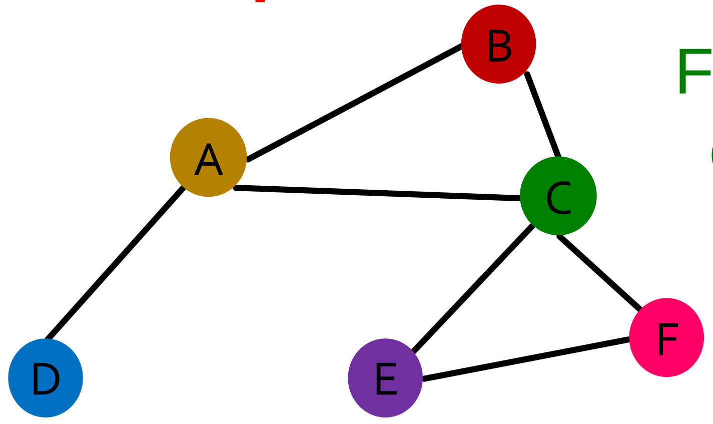
What does it mean by "graph representation is same for two order plans"?

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $\mathbb{R}^d$ then

$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) = f(\boldsymbol{A}_2, \boldsymbol{X}_2)$$

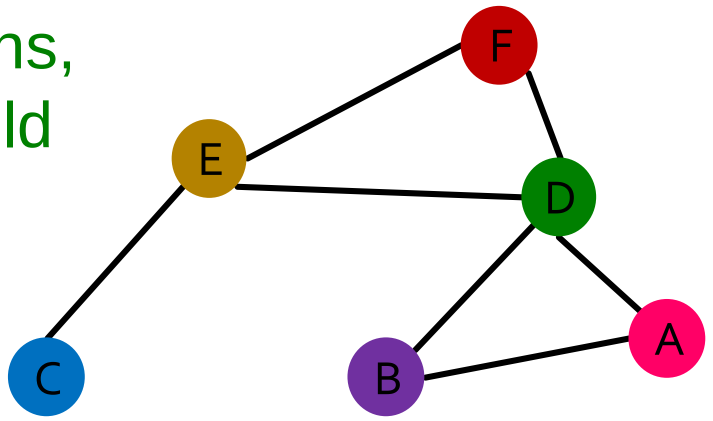
$\boldsymbol{A}$ is the adjacency matrix
$\boldsymbol{X}$ is the node feature matrix

Order plan 1: A_1, X_1



For two order plans, output of f should be the same!

Order plan 2: A_2, X_2

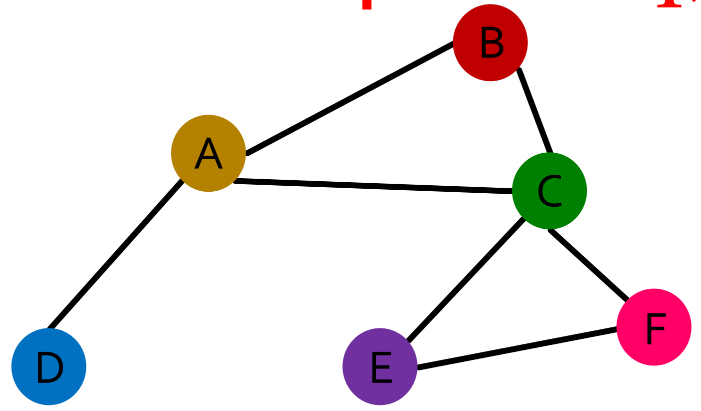


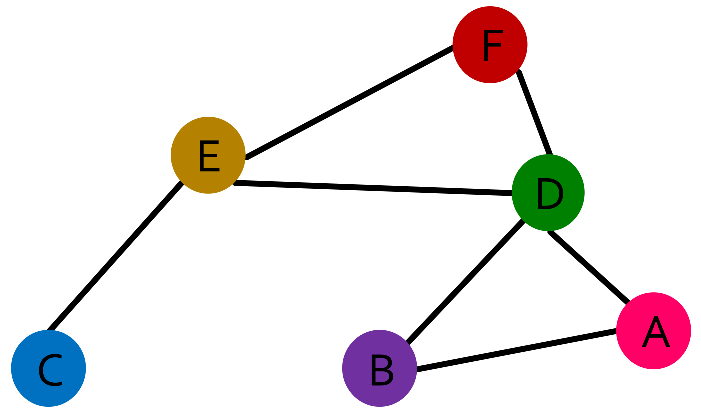
Recap: Permutation Equivariance

For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{m\times d}$.

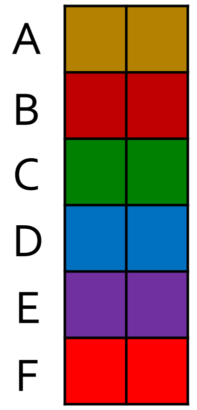
Order plan 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$

Order plan 2: $\boldsymbol{A}_2, \boldsymbol{X}_2$

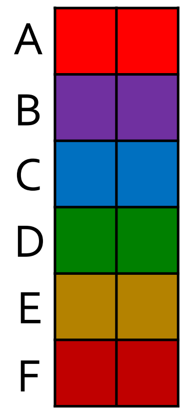




$f(\boldsymbol{A}_1, \boldsymbol{X}_1) =$

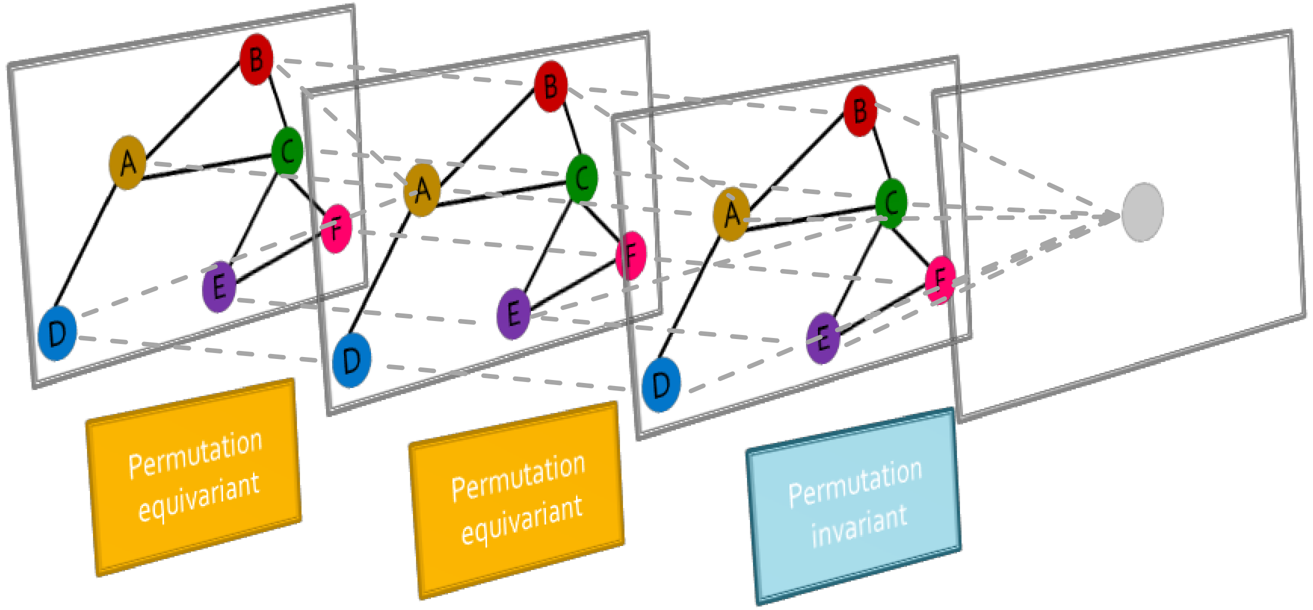


$f(\boldsymbol{A}_2, \boldsymbol{X}_2) =$



Recap: Graph Neural Networks Overview

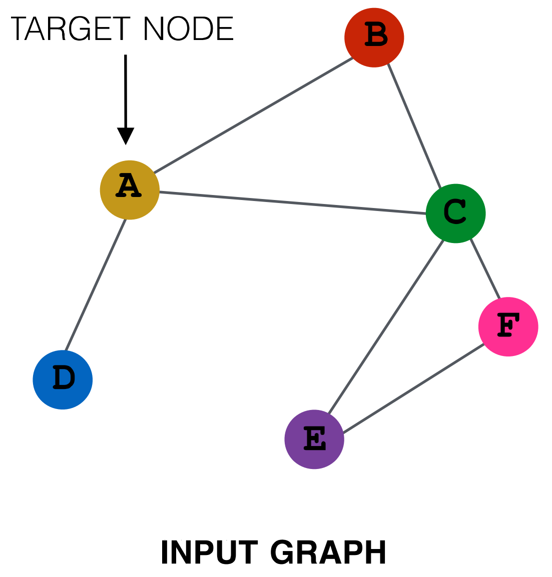
- GNNs consist of multiple permutation equivariant / invariant functions

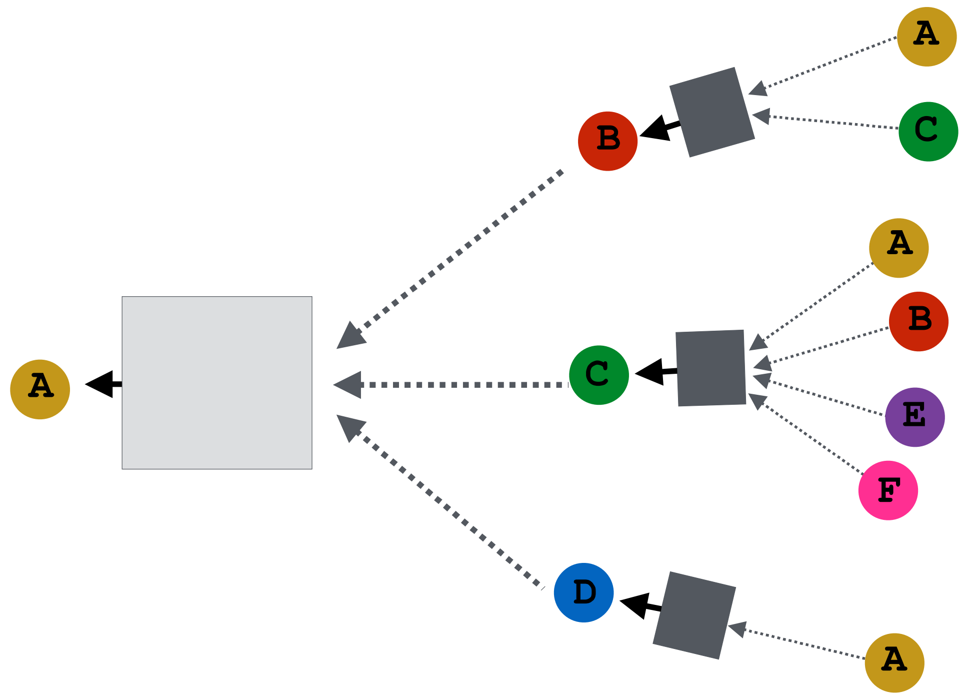


- Next: Design GNNs that are permutation equivariant / invariant by **passing and aggregating information from neighbors**

Idea: Aggregate Neighbors

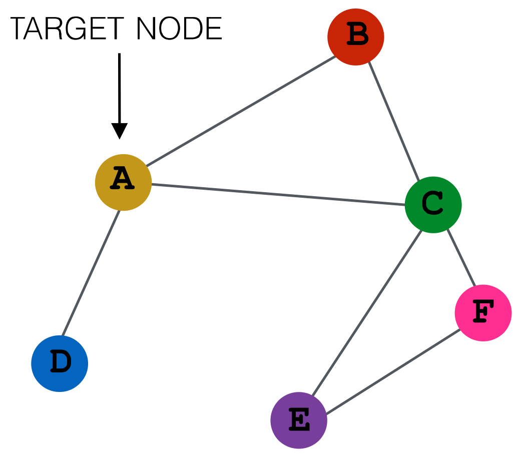
- **Key idea:** Generate node embeddings based on **local network neighborhoods**



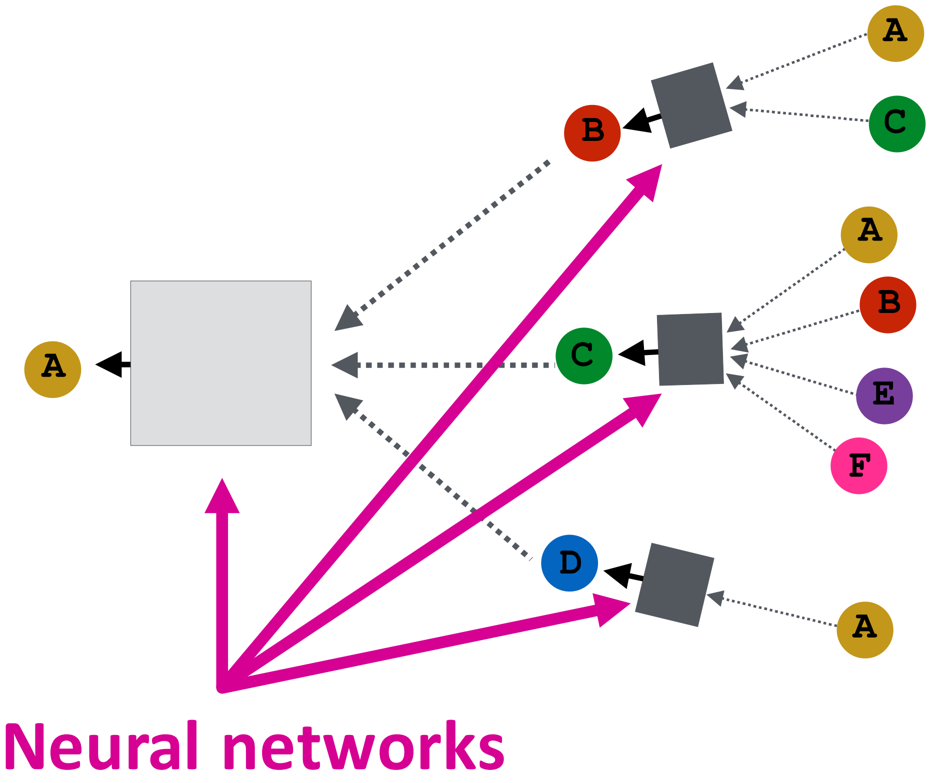


Idea: Aggregate Neighbors

- **Intuition:** Nodes aggregate information from their neighbors using neural networks

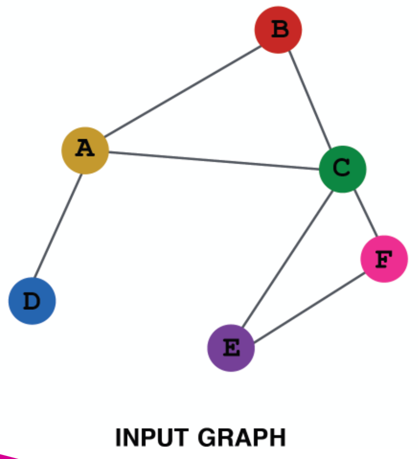


INPUT GRAPH

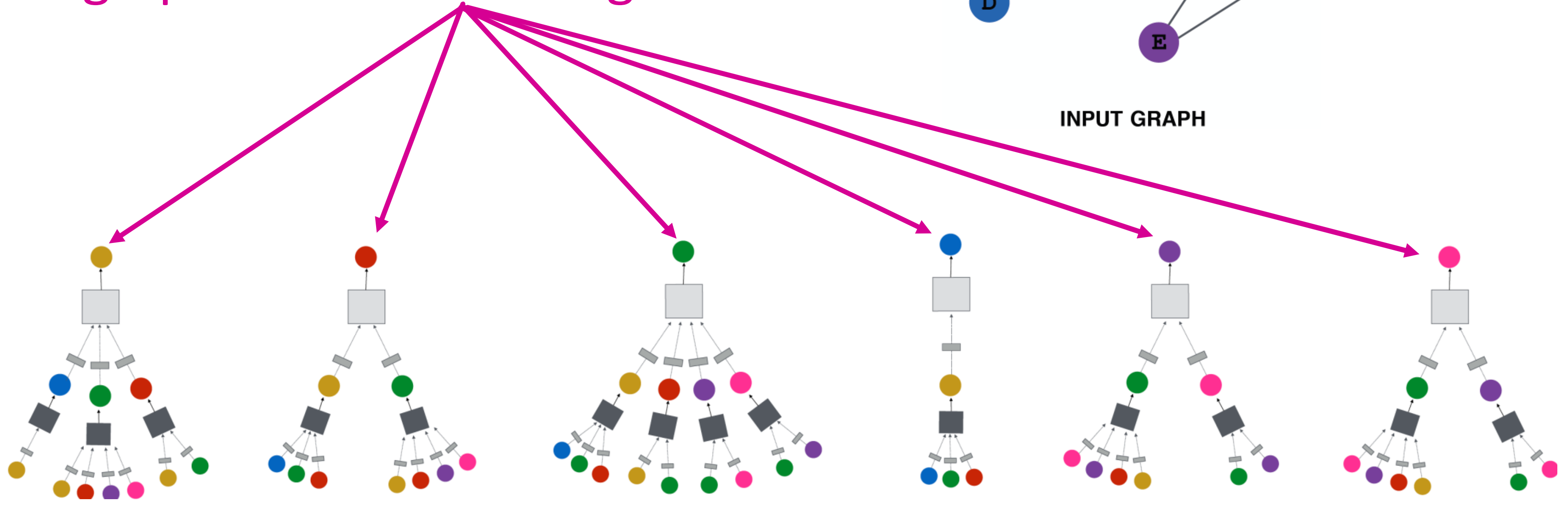


Idea: Aggregate Neighbors

- **Intuition:** Network neighborhood defines a computation graph

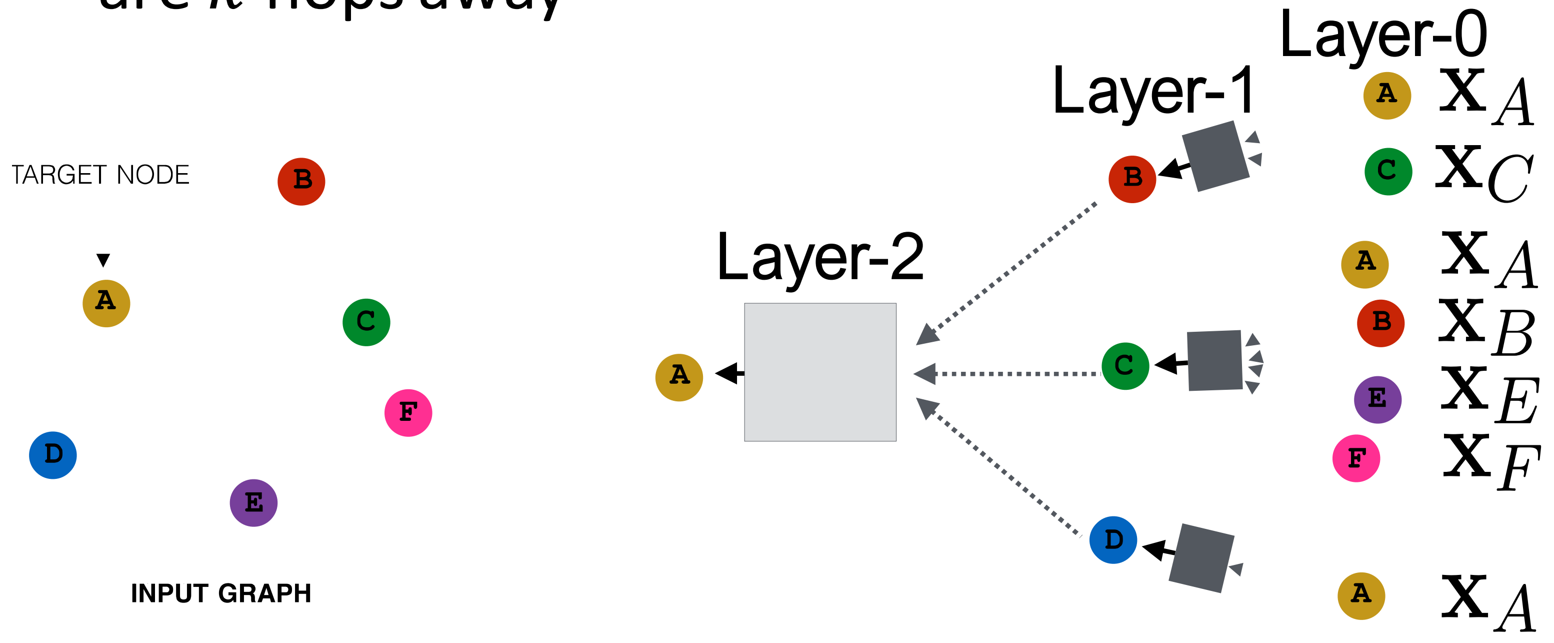


Every node defines a computation graph based on its neighborhood!



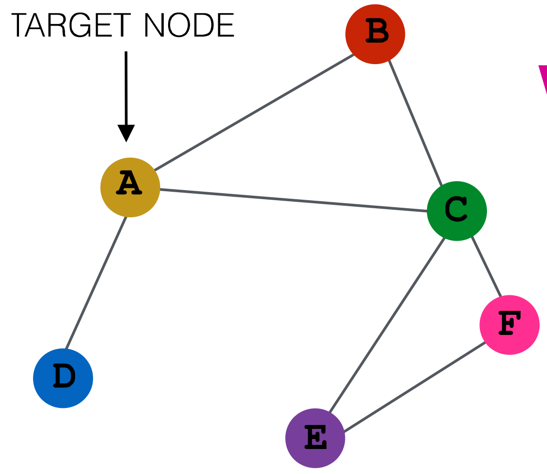
Deep Model: Many Layers

- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node v is its input feature, x_v
  - Layer-k embedding gets information from nodes that are k hops away



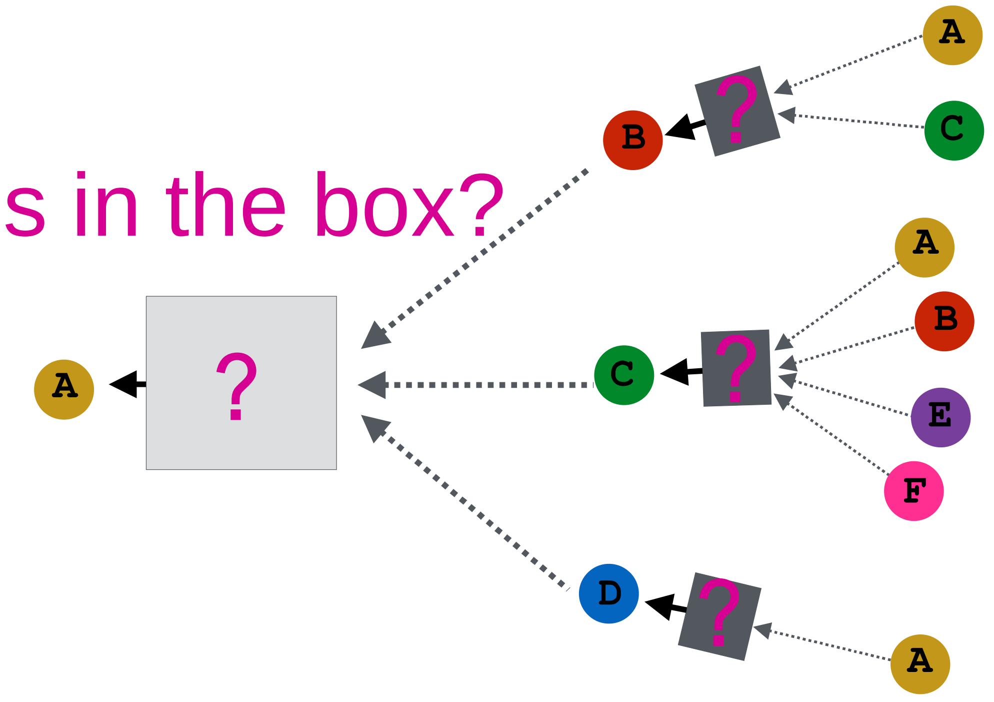
Neighborhood Aggregation

- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



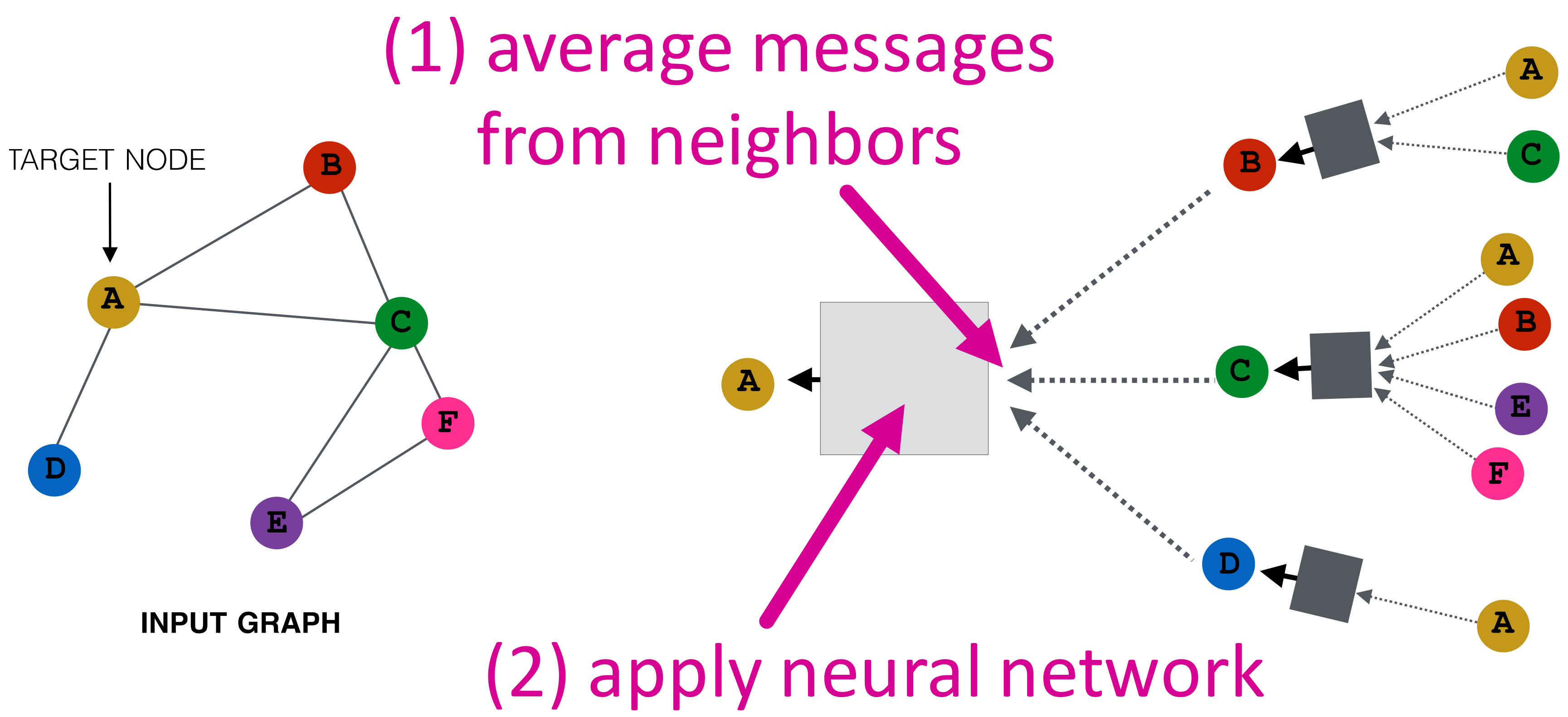
INPUT GRAPH

What is in the box?



Neighborhood Aggregation

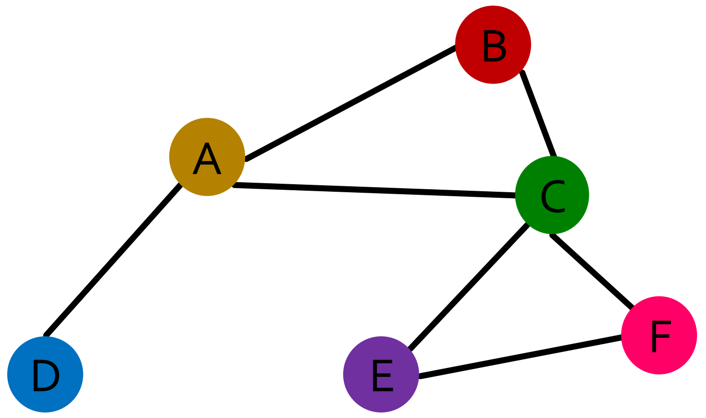
- **Basic approach:** Average information from neighbors and apply a neural network



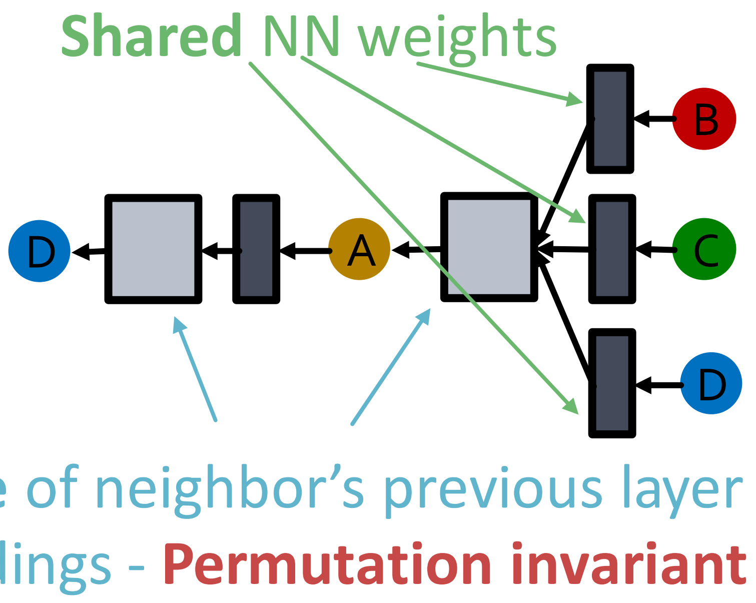
GCN (Graph Convolutional Net): Invariance and Equivariance

What are the invariance and equivariance properties for a GCN?

- **Given a node**, the GCN that computes its embedding is **permutation invariant**



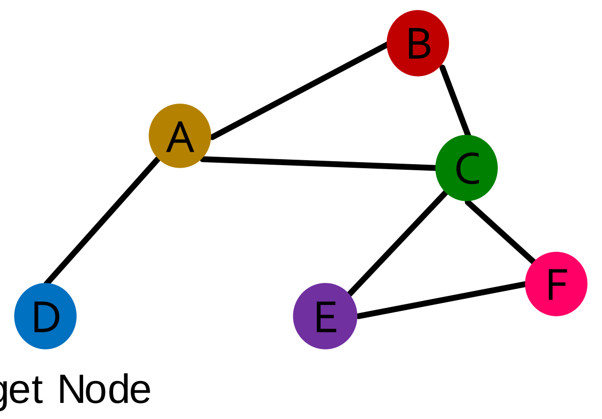
Target Node

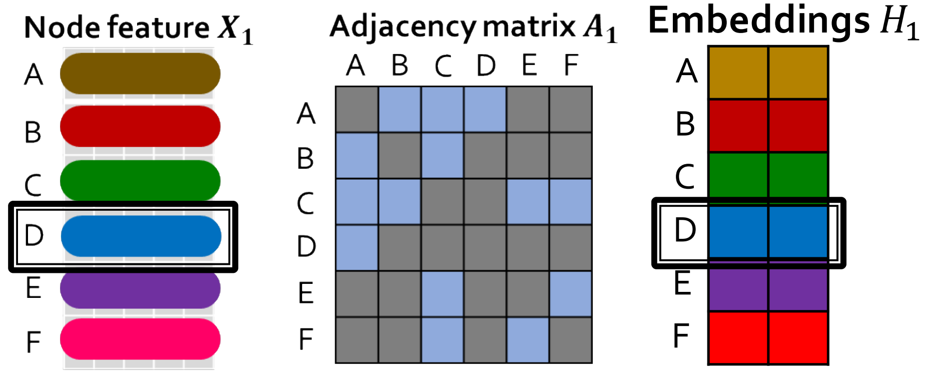

Shared NN weights
D
A
B
C
D
Average of neighbor's previous layer embeddings - **Permutation invariant**

GCN: Invariance and Equivariance

- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**

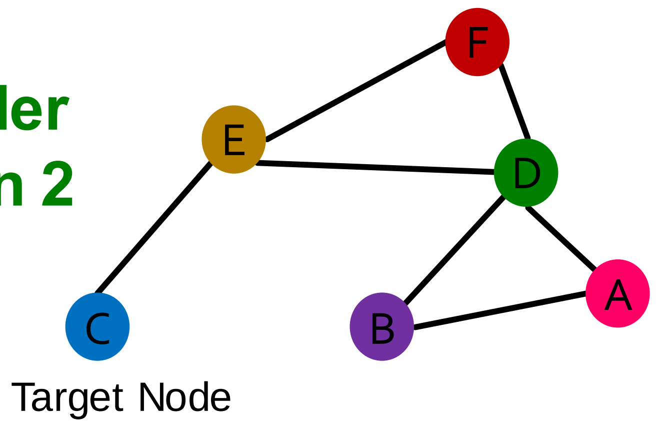
Order plan 1

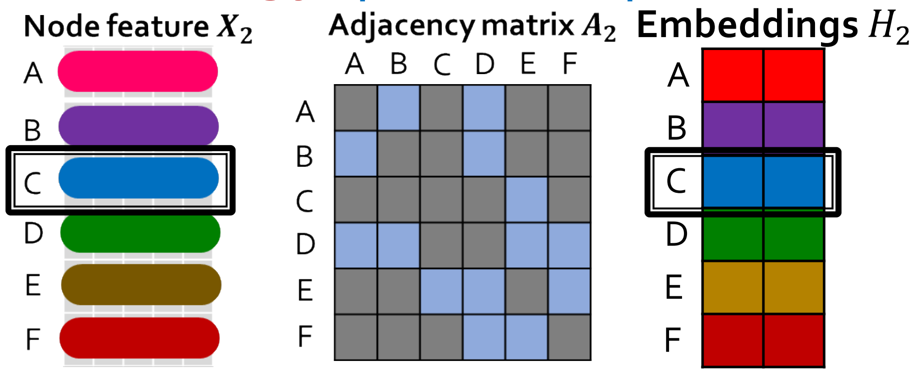




Permute the input, the output also permutes accordingly - permutation equivariant

Order plan 2





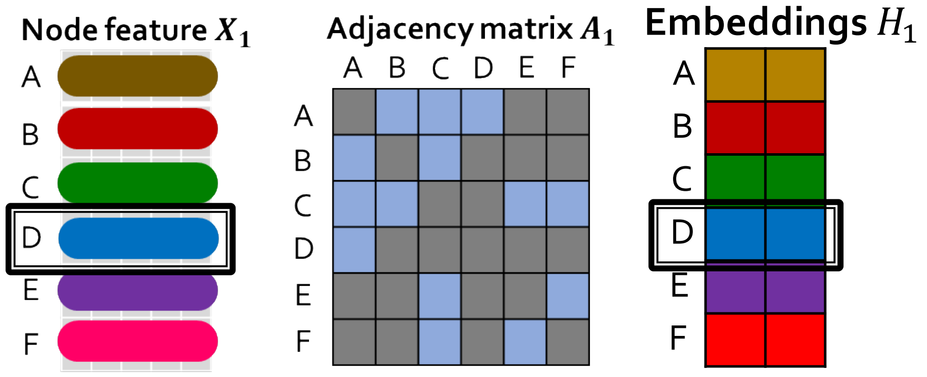
GCN: Invariance and Equivariance

- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**

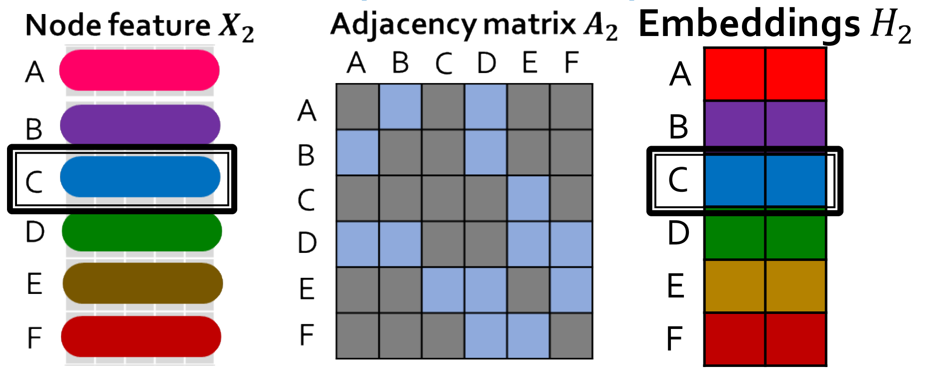
Detailed reasoning:

1. The rows of **input node features** and **output embeddings** are **aligned**
2. We know computing the embedding of **a given node** with GCN is **invariant.**
3. So, after permutation, the **location** of **a given node** in the **input node feature** matrix is changed, and the **the output embedding of a given node stays the same** (the colors of node feature and embedding are **matched**)

This is permutation equivariant

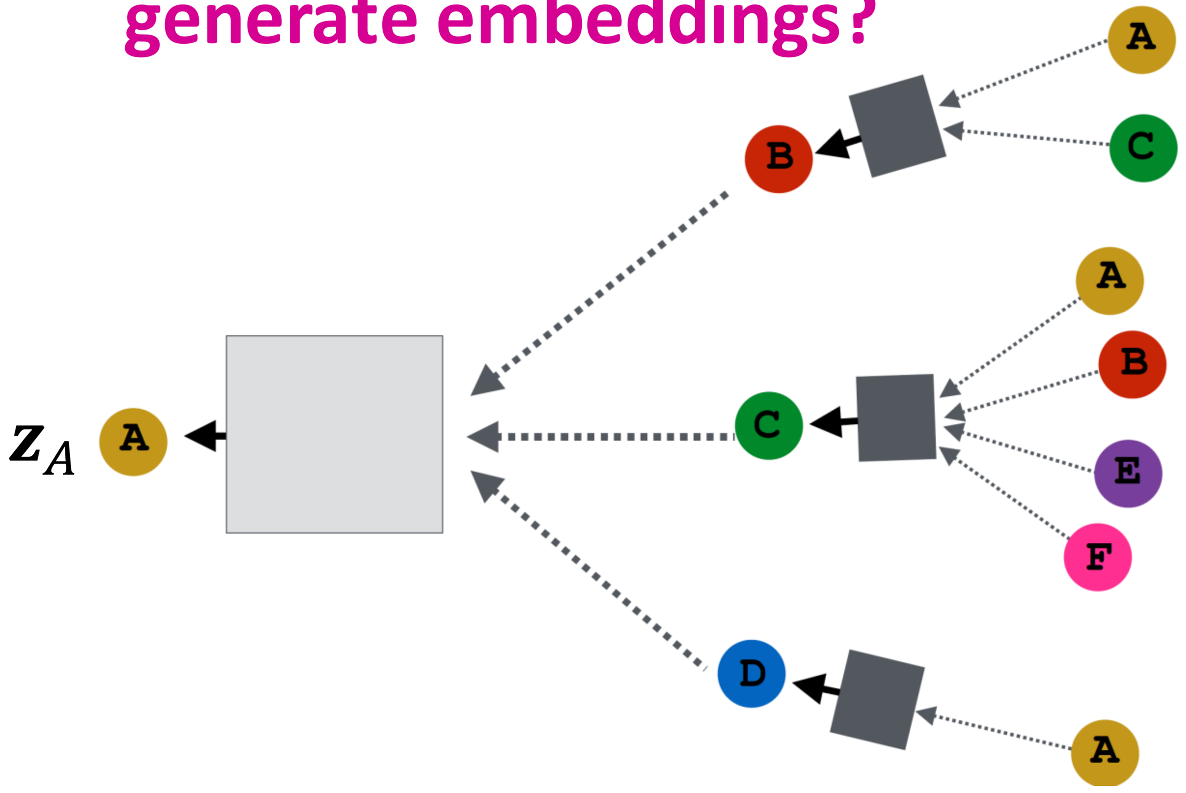


Permute the input, the output also permutes accordingly - permutation equivariant



How to Train A GNN

How do we train the GCN to generate embeddings?



Need to define a loss function on the embeddings.

How to Train A GNN

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised setting**: we want to minimize the loss $\mathcal{L}$ (see also Slide 15):

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical

How to Train A GNN

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised setting**: we want to minimize the loss $\mathcal{L}$ (see also Slide 15):

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

How to Train A GNN

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised setting**: we want to minimize the loss $\mathcal{L}$ (see also Slide 15):

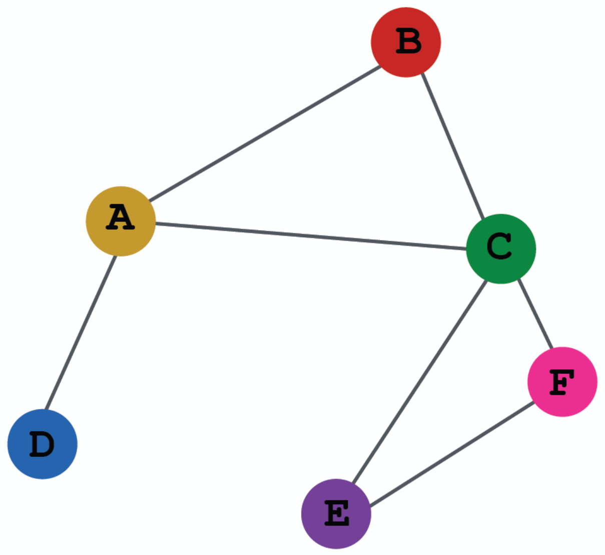
$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

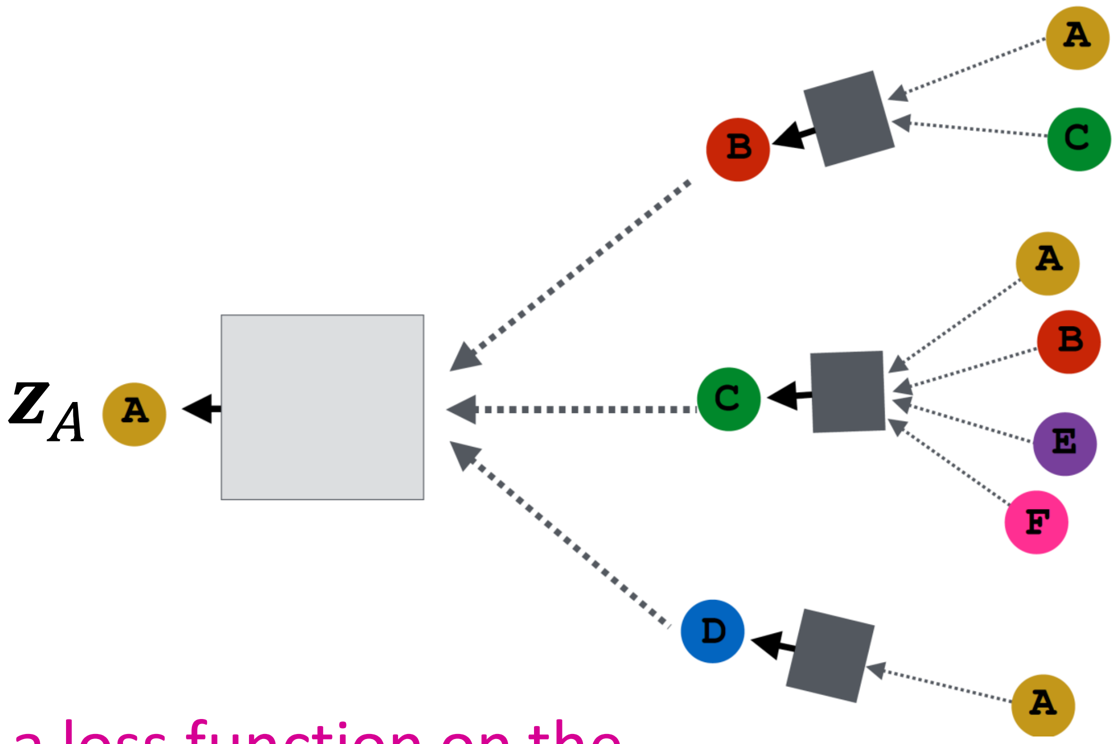
  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

"Similar" nodes have similar embeddings (discussed in last lecture)

Model Design: Overview

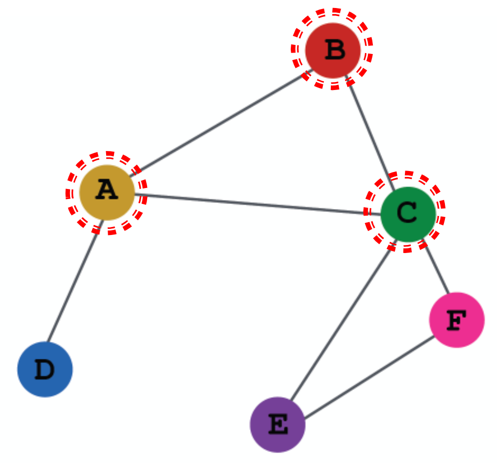
(1) Define a neighborhood aggregation function





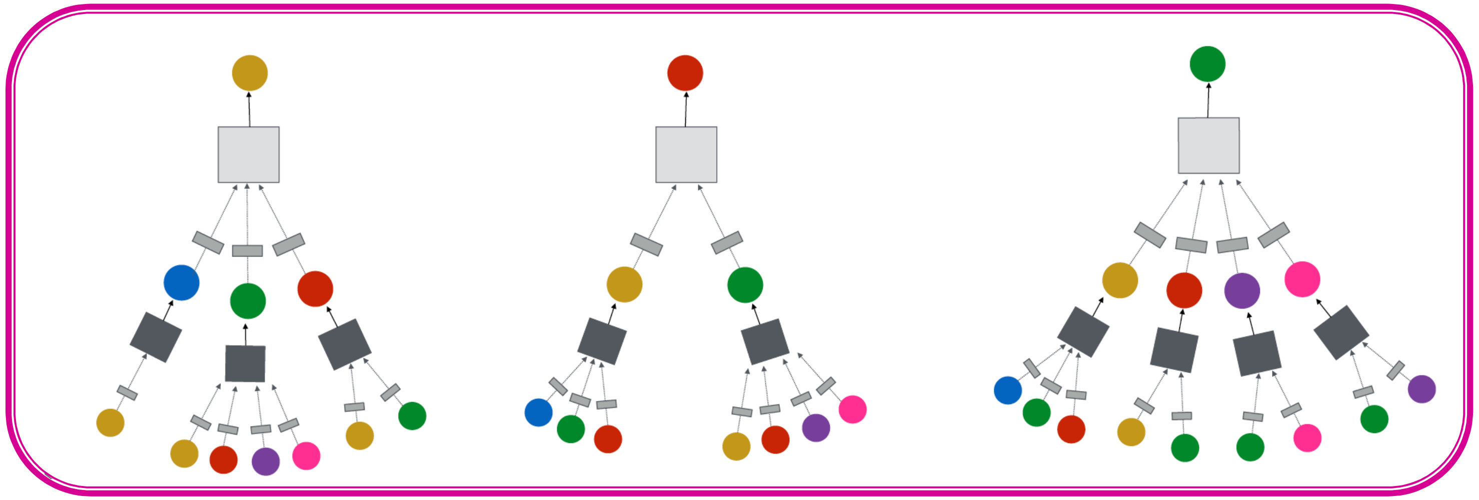
(2) Define a loss function on the embeddings

Model Design: Overview

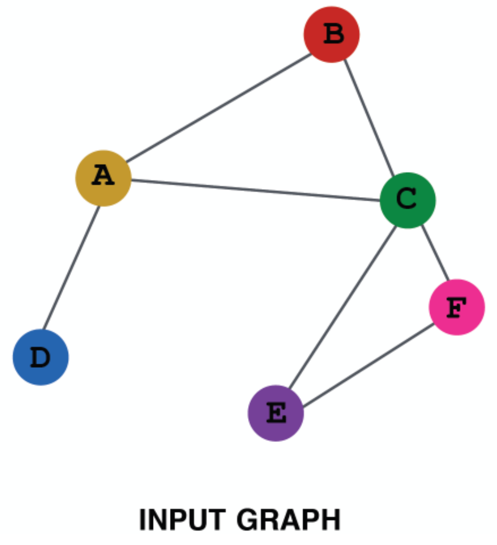


INPUT GRAPH

(3) Train on a set of nodes, i.e., a batch of compute graphs

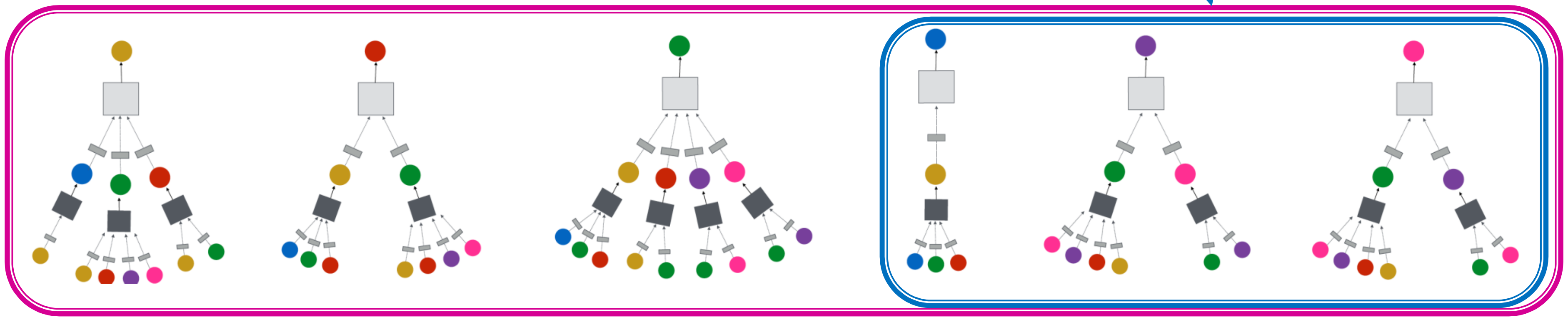


Model Design: Overview



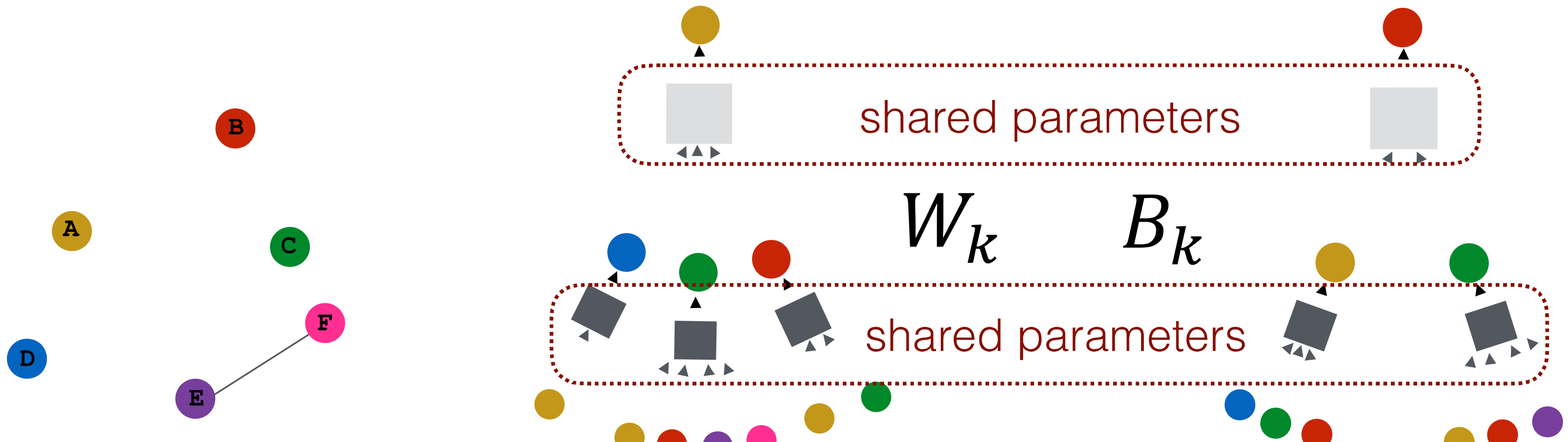
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!



Inductive Capability

- **The same aggregation parameters are shared for all nodes:**
  - The number of model parameters is sublinear in $|V|$ and we can **generalize to unseen nodes**!

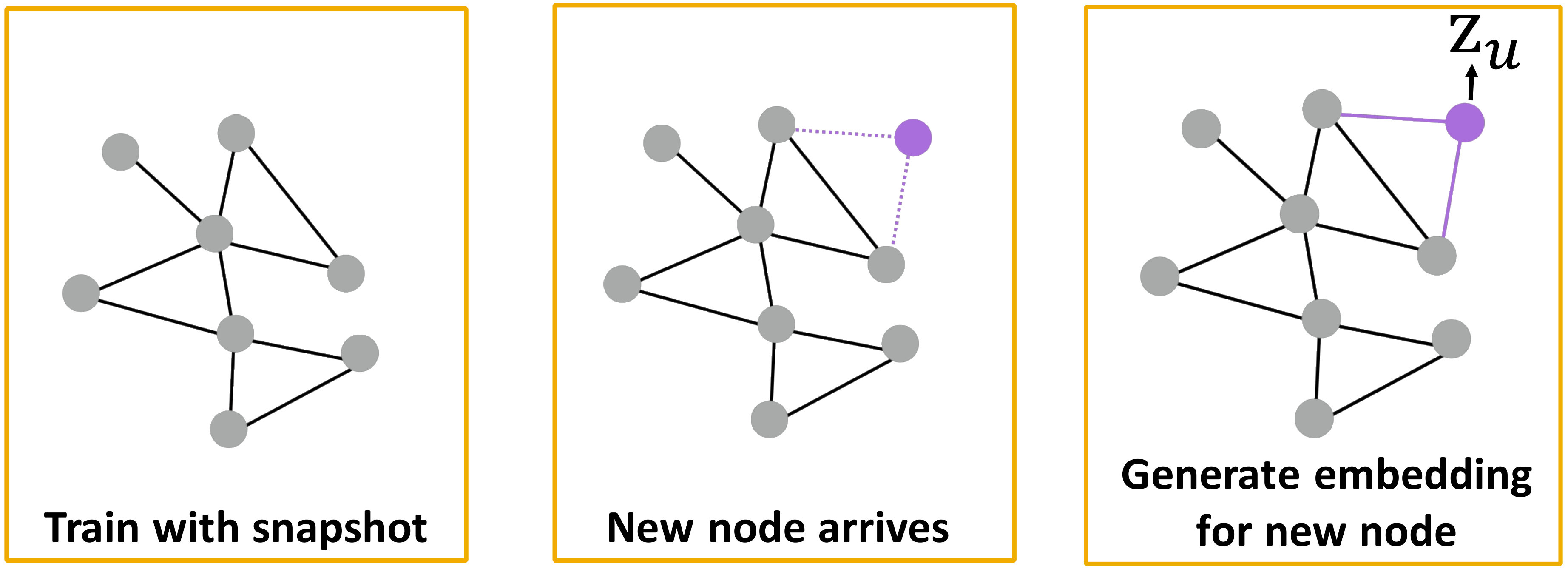


INPUT GRAPH

Compute graph for node A

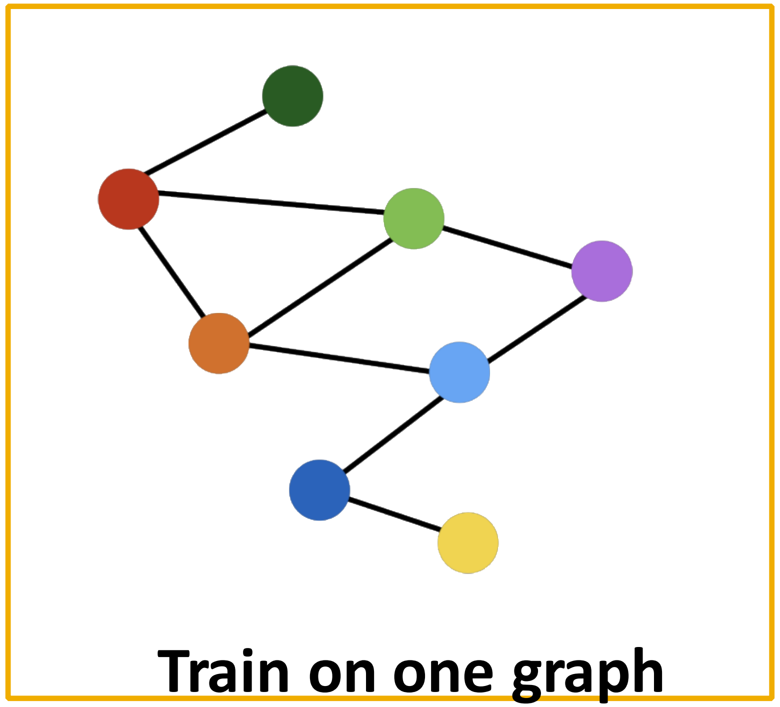
Compute graph for node B

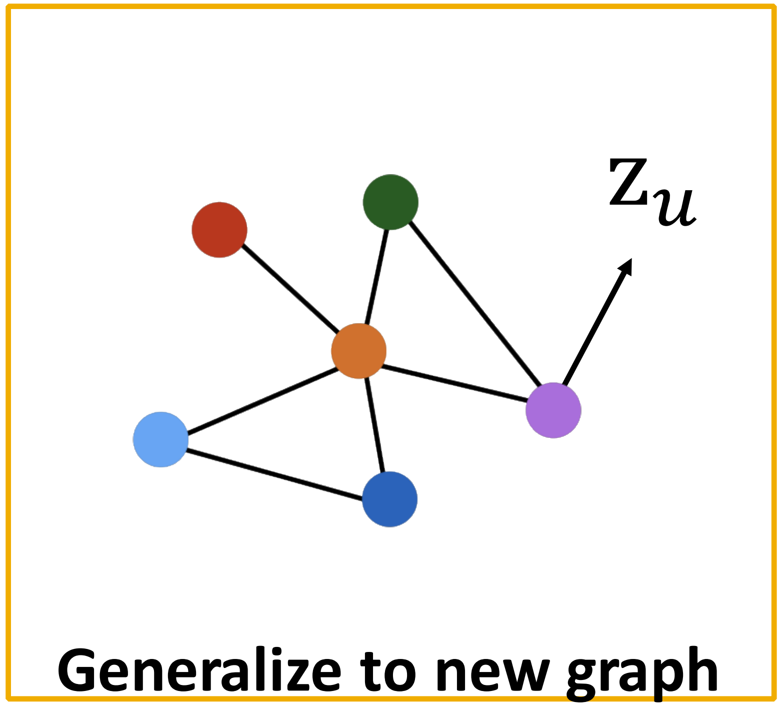
Inductive Capability: New Nodes



- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

Inductive Capability: New Graphs

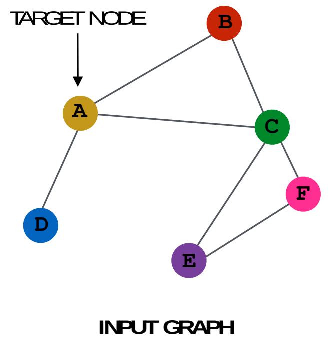




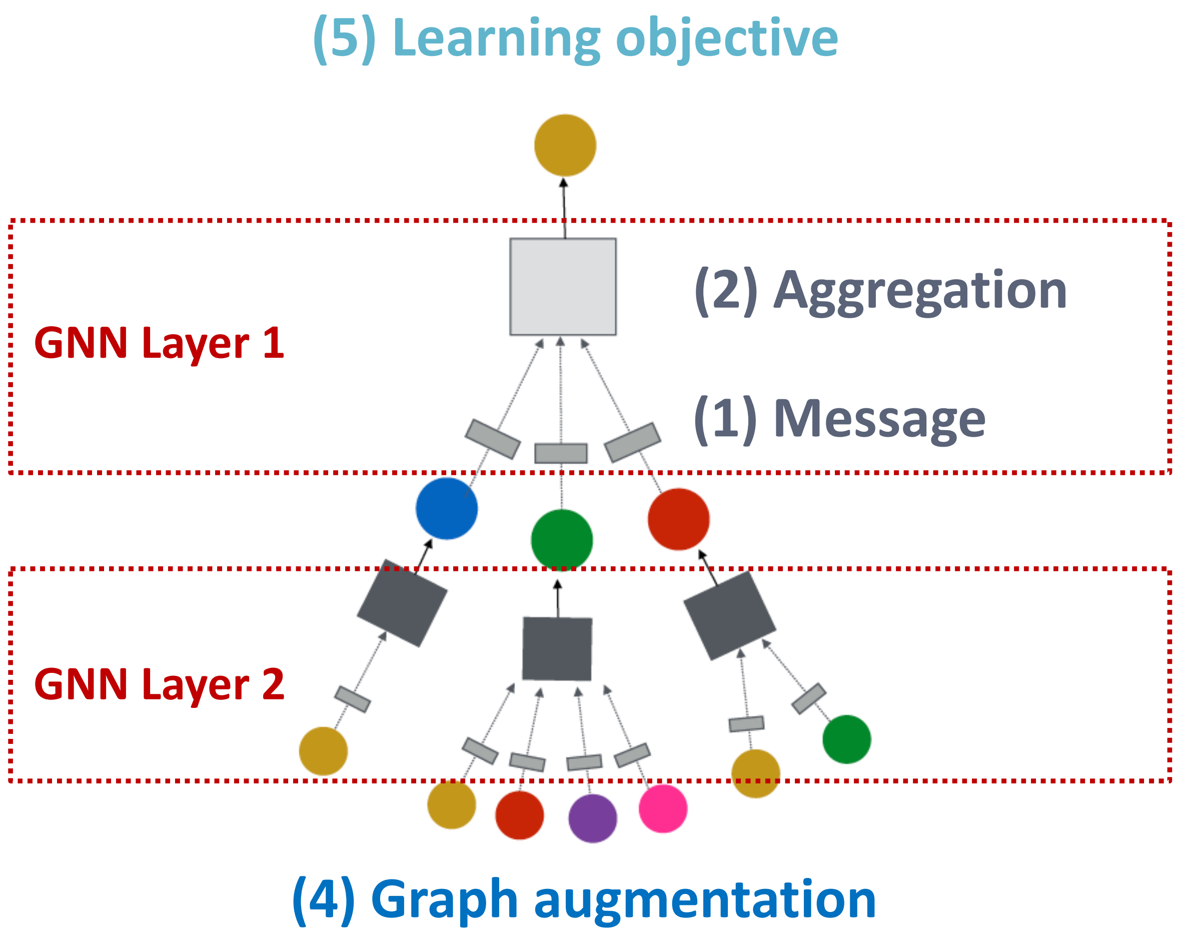
Inductive node embedding $\longrightarrow$ Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Discussion: Design Space of GNNs



(3) Layer connectivity



Ex1: Connectivity

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Feature level:**
  - The input graph **lacks features** → feature augmentation
- **Structure level:**
  - The graph is **too sparse** à inefficient message passing
  - The graph is **too dense** à message passing is too costly
  - The graph is **too large** à cannot fit the computational graph into a GPU
- It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

Ex1: Connectivity

- **Graph Feature manipulation**
  - The input graph **lacks features** → **feature augmentation**
- **Graph Structure manipulation**
  - The graph is **too sparse** → **Add virtual nodes / edges**
  - The graph is **too dense** → **Sample neighbors when doing message passing**
  - The graph is **too large** → **Sample subgraphs to compute embeddings**
    - Will cover later in lecture: Scaling up GNNs

Ex2: Graph Attention Network (GAT)

¡ **In GCN**

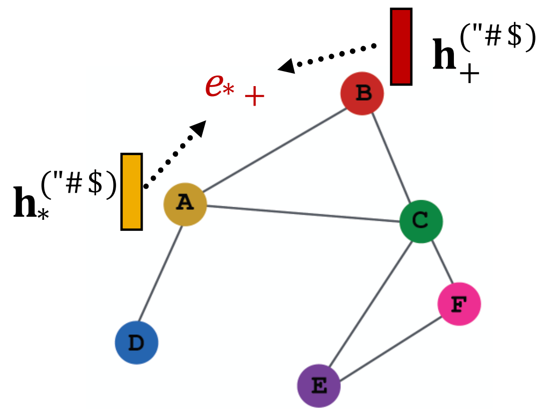
§ $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v

§ $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)

§ $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Not all node's neighbors are equally important

- Query, Key, Value
- Alignment $\boldsymbol{e}$
- $\boldsymbol{a} = \mathrm{softmax}(\boldsymbol{e})$



Knowledge Graphs (KGs)

Slides adapted from:

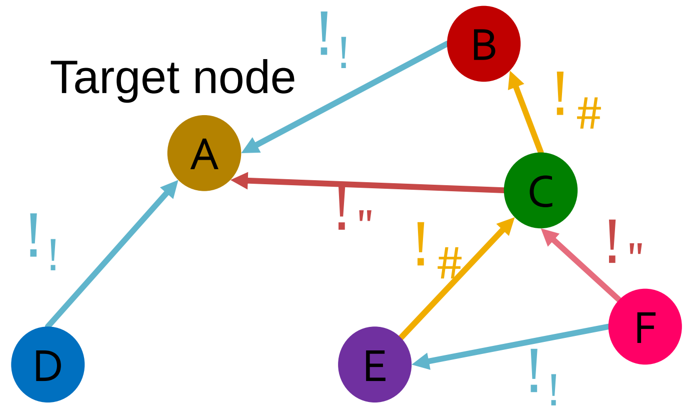
- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs

Outline

- Overview
- Knowledge Graph Completion (Link Prediction)
- Reasoning on Knowledge Graphs

Heterogeneous Graphs

- **Heterogeneous graphs: a graph with multiple relation types**



Input graph

Heterogeneous Graphs

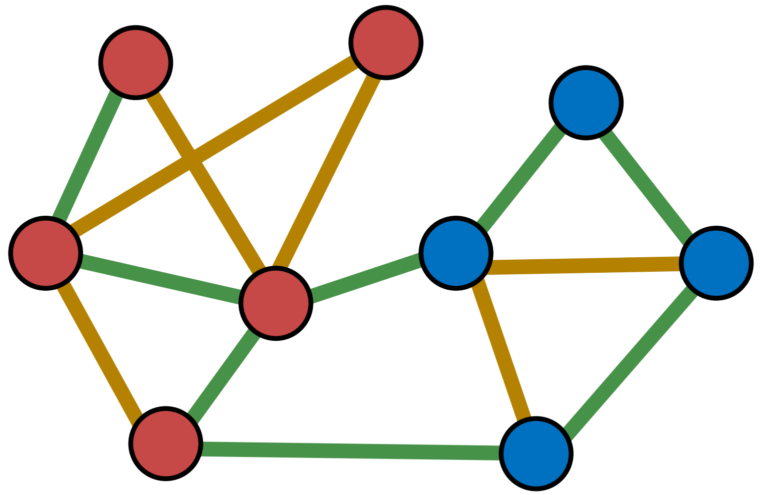
8 possible relation types!

(Paper, Cite, Paper)

(Paper, Like, Paper)

(Paper, Cite, Author)

(Paper, Like, Author)



(Author, Cite, Author)

(Author, Like, Author)

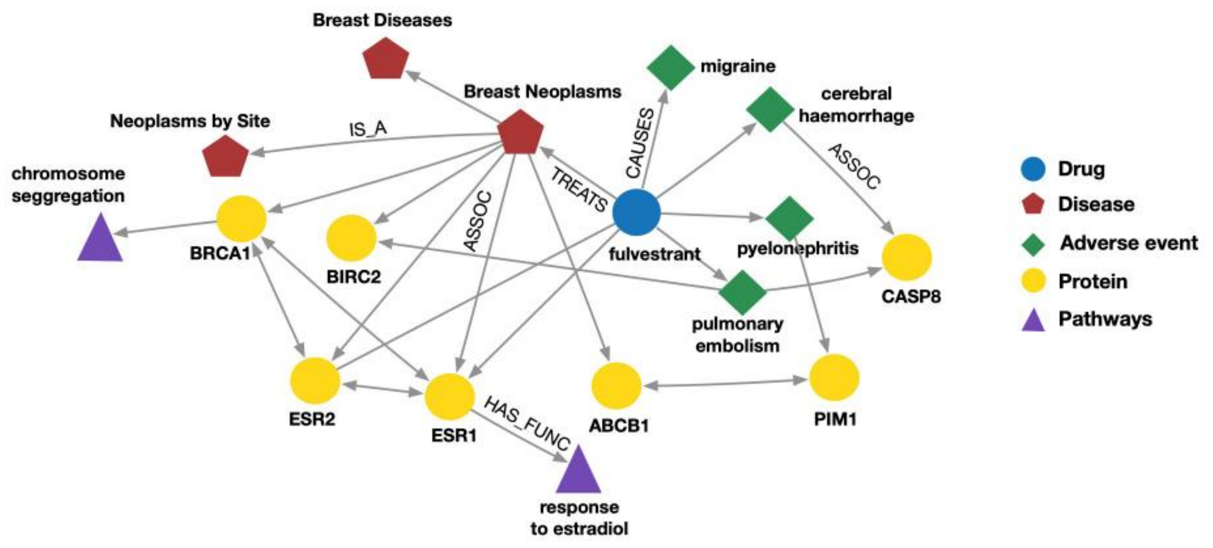
(Author, Cite, Paper)

(Author, Like, Paper)

Relation types: (node_start, edge, node_end)

- We use **relation type to describe an edge** (as opposed to edge type)
- Relation type better captures the interaction between nodes and edges

Heterogeneous Graphs



Biomedical Knowledge Graphs

Example node: Migraine

Example relation: (fulvestrant, Treats, Breast Neoplasms)

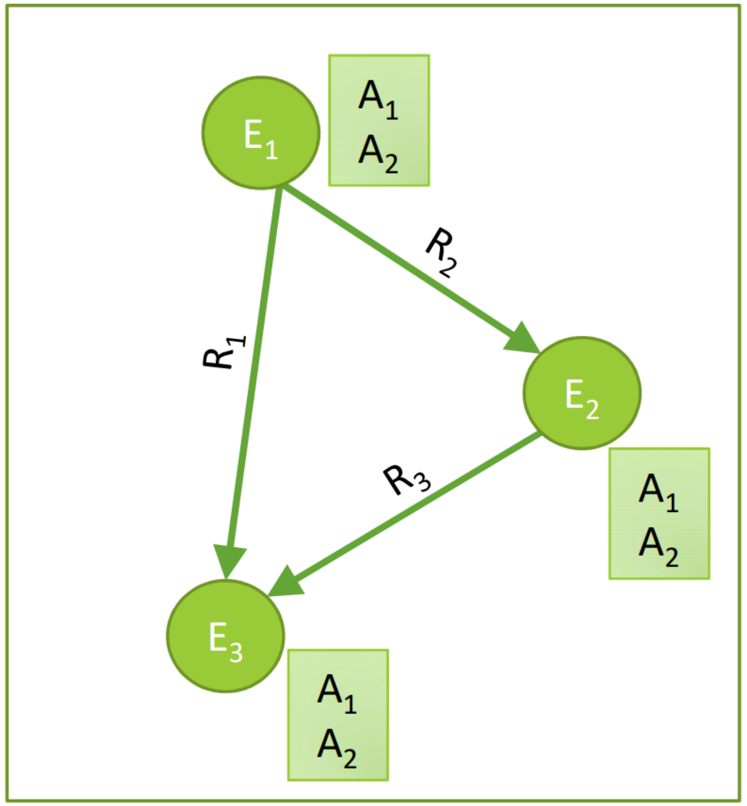
Example node type: Protein

Example edge type: Causes

Knowledge Graph

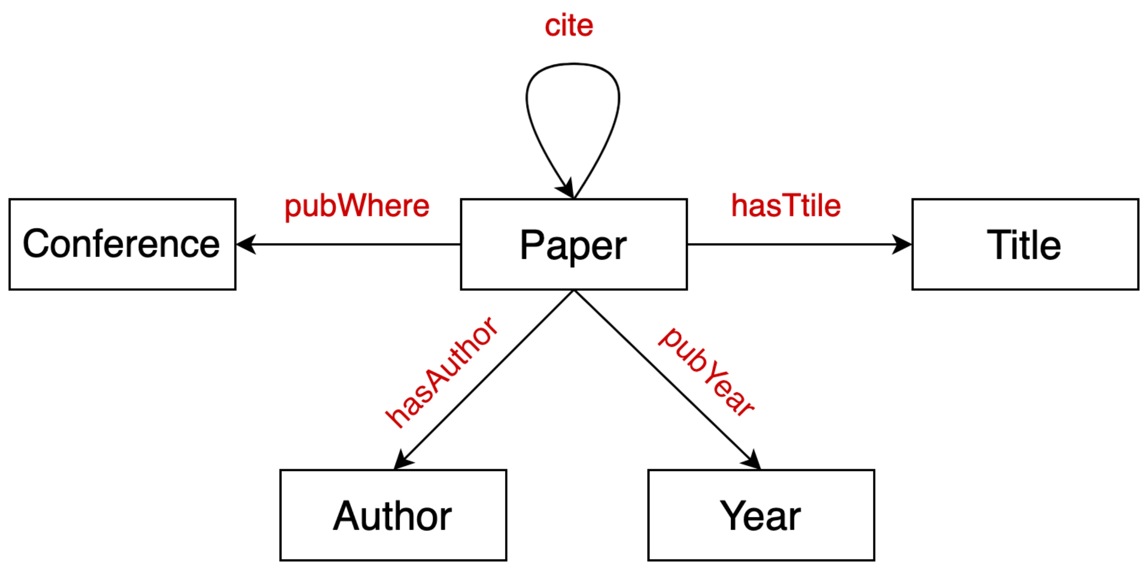
Knowledge in graph form:

- Capture entities, types, and relationships
- Nodes are **entities**
- Nodes are labeled with their **types**
- Edges between two nodes capture **relationships** between entities
- **KG is an example of a heterogeneous graph**



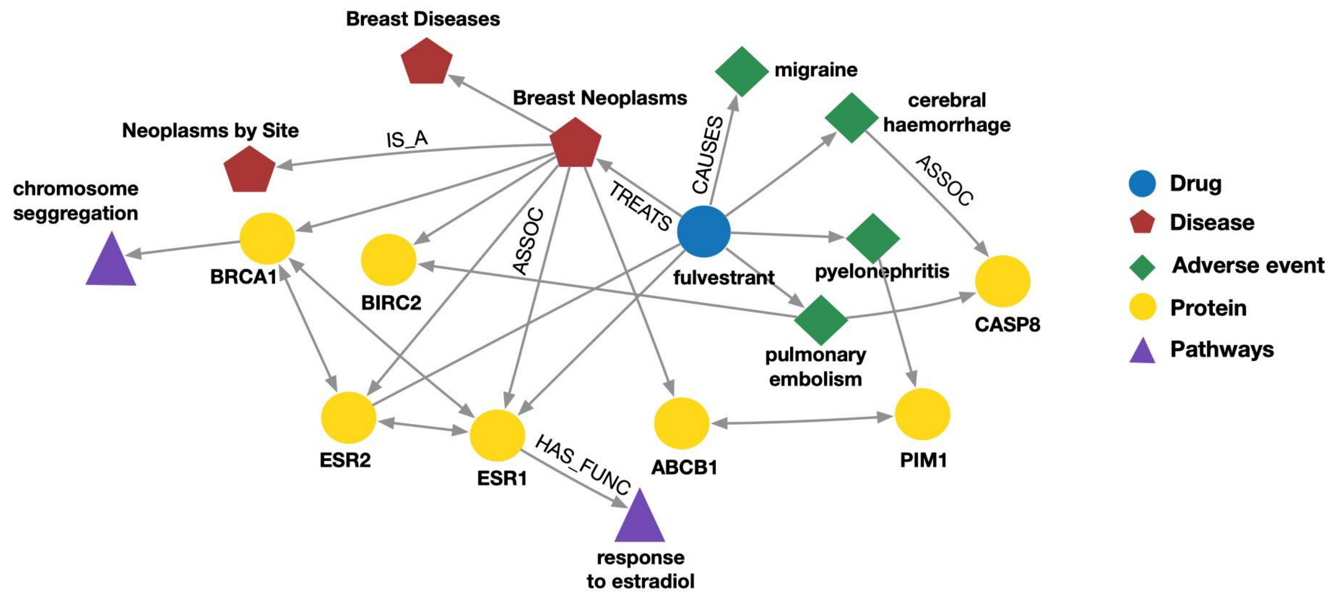
Example: Bibliographic Networks

- **Node types**: paper, title, author, conference, year
- **Relation types**: pubWhere, pubYear, hasTitle, hasAuthor, cite



Example: Bio Knowledge Graphs

- **Node types**: drug, disease, adverse event, protein, pathways
- **Relation types**: has_func, causes, assoc, treats, is_a



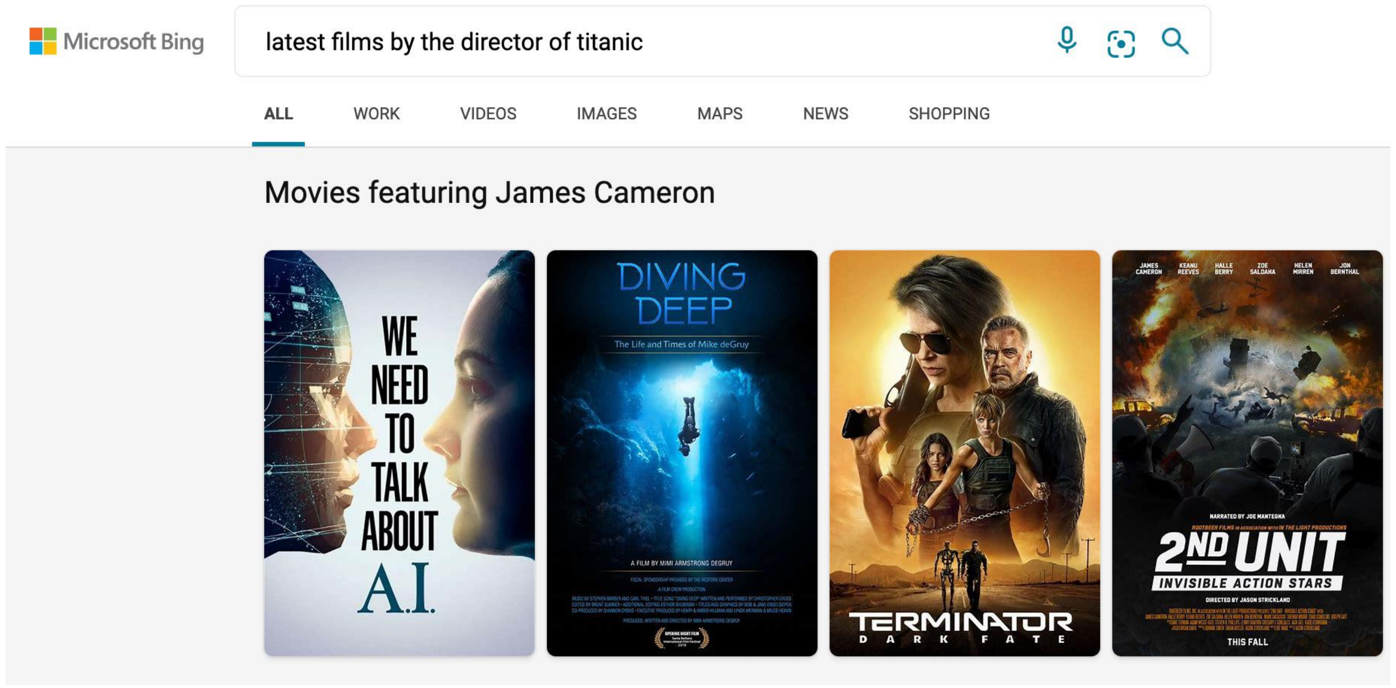
KGs in Practice

Examples of knowledge graphs

- Google Knowledge Graph
- Amazon Product Graph
- Facebook Graph API
- IBM Watson
- Microsoft Satori
- Project Hanover/Literome
- LinkedIn Knowledge Graph
- Yandex Object Answer

Applications of KGs

- **Serving information:**



Microsoft Bing

latest films by the director of titanic

ALL WORK VIDEOS IMAGES MAPS NEWS SHOPPING

Movies featuring James Cameron

Image credit: Bing

Applications of KGs

- **Question answering and conversation agents**

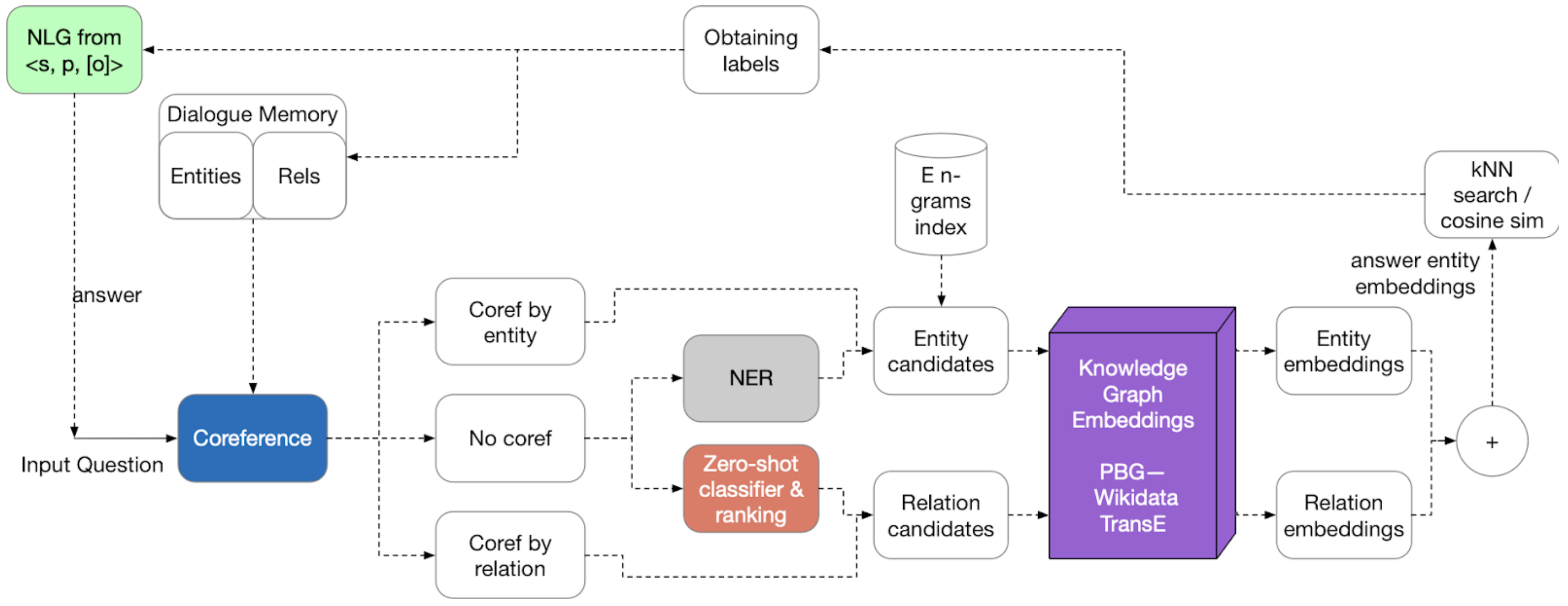


Image credit: Medium

KG Datasets

- **Publicly available KGs:**
  - FreeBase, Wikidata, Dbpedia, YAGO, NELL, etc.
- **Common characteristics:**
  - **Massive**: Millions of nodes and edges
  - **Incomplete**: Many true edges are missing

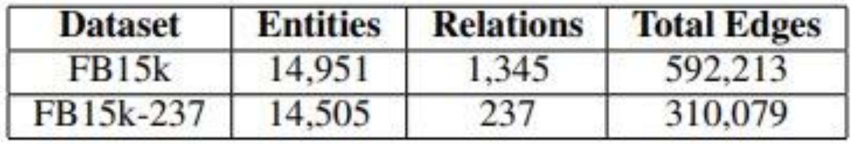
Given a massive KG, enumerating all the possible facts is intractable! $\Rightarrow$ **Can we predict plausible BUT missing links?**

Example: Freebase

- ¡ **Freebase**
  - § ~80 million **entities**
  - § ~38K **relation types**
  - § ~3 billion **facts/triples**

93.8% of persons from Freebase have no place of birth and 78.5% have no nationality!

- ¡ **Datasets:** FB15k/FB15k-237
  - § A **complete** subset of Freebase, used by researchers to learn KG models



Dataset	Entities	Relations	Total Edges
FB15k	14,951	1,345	592,213
FB15k-237	14,505	237	310,079

[1] Paulheim, Heiko. "Knowledge graph refinement: A survey of approaches and evaluation methods." *Semantic web* 8.3 (2017): 489-508.
[2] Min, Bonan, et al. "Distant supervision for relation extraction with an incomplete knowledge base." *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2013.

Outline

- Overview
- **Knowledge Graph Completion (Link Prediction)**
- Reasoning on Knowledge Graphs

Questions?