

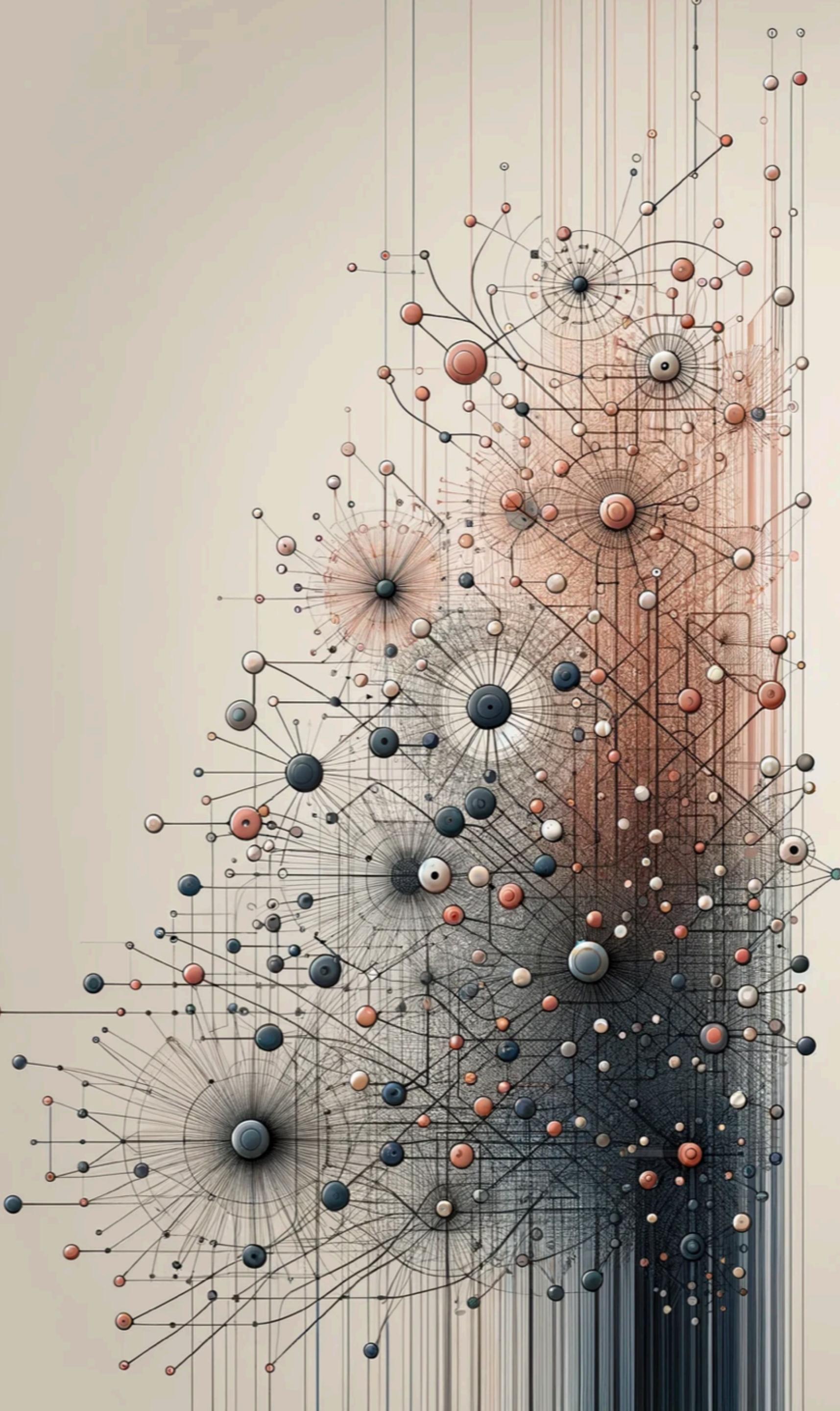


UNIVERSITÀ
DI TORINO

Analisi e Visualizzazione delle Reti Complesse

**NS22 - Traditional Machine Learning
on Graphs**

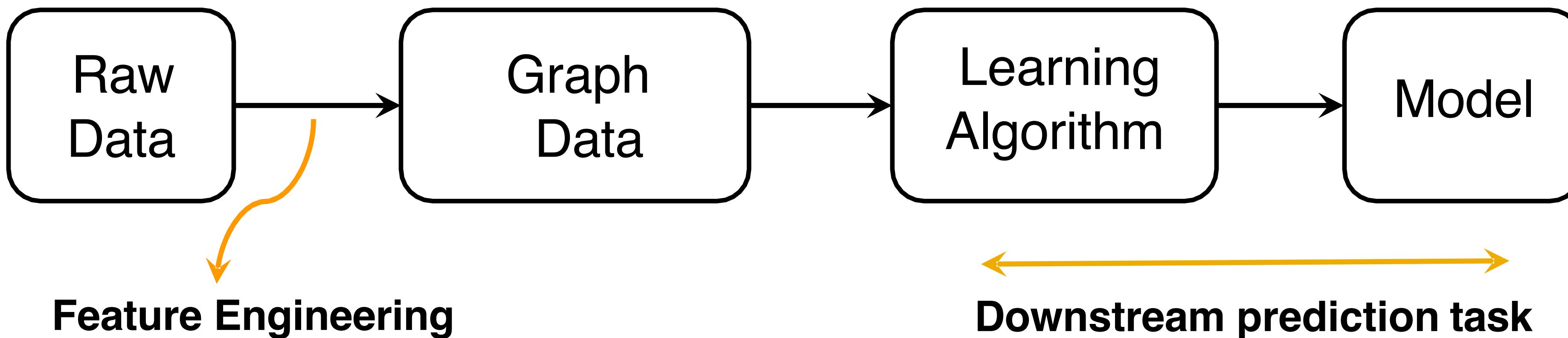
Prof. Rossano Schifanella



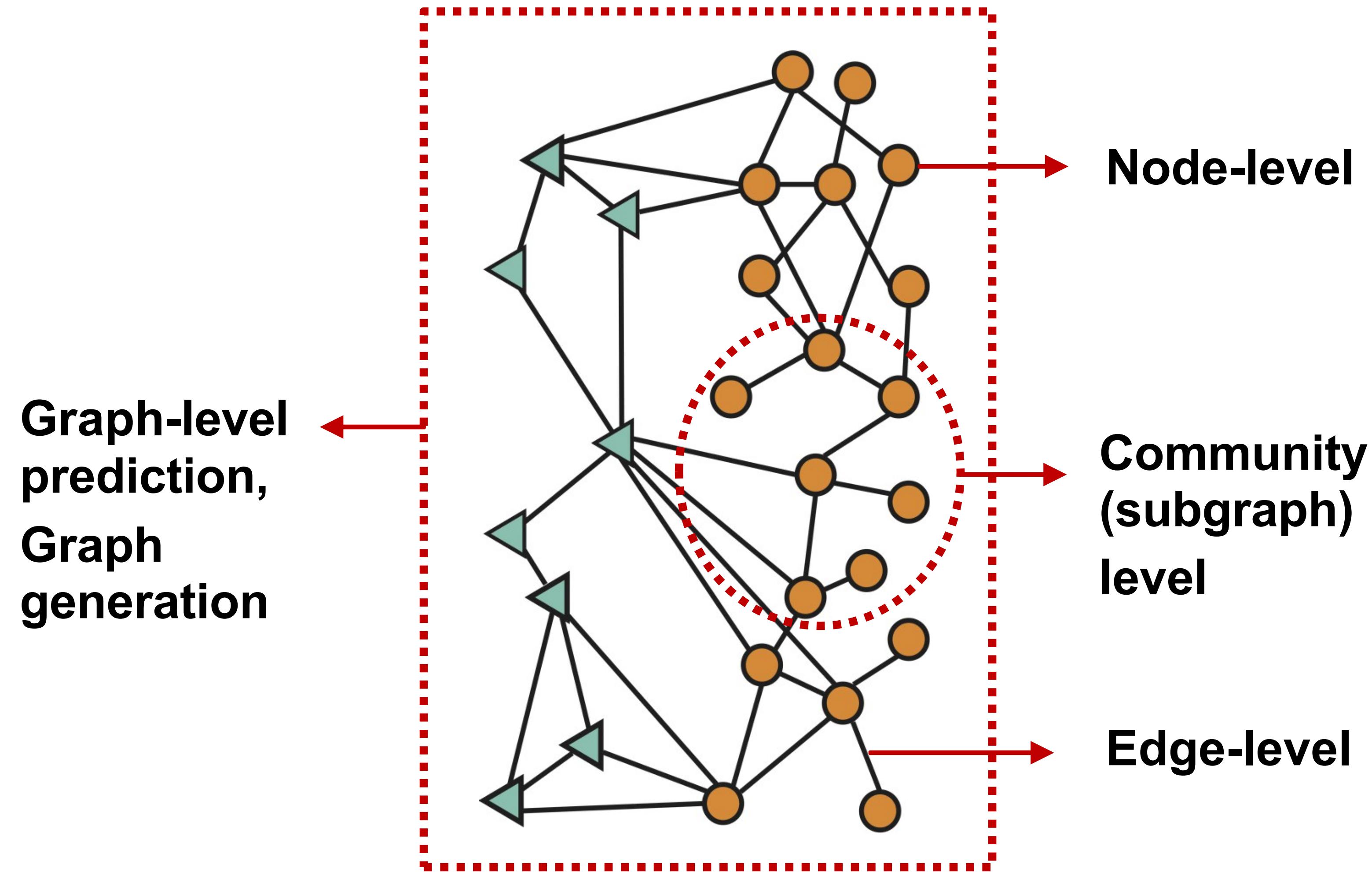
General Pipeline

(Supervised) Machine Learning Lifecycle

This feature, that feature. **Every single time!**



Different types of tasks



Classic Graph ML Tasks

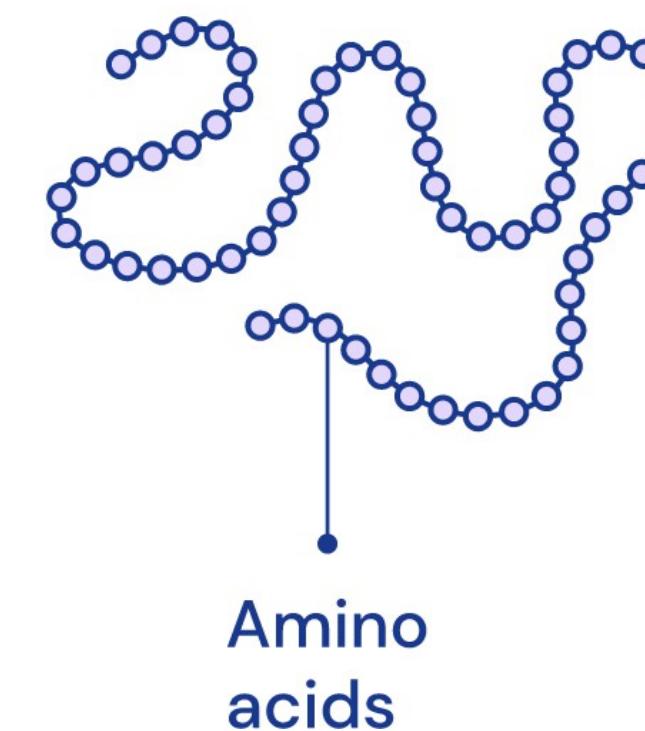
- **Node classification:** predict a property of a node
 - Example: Categorize online users/items
- **Link prediction:** predict whether there are missing links between two nodes
 - Example: Knowledge graph completion
- **Graph classification:** categorize different graphs
 - Example: Molecule property prediction
- **Clustering:** detect if nodes form a community
 - Example: Social circle detection
- **Other tasks:**
 - **Graph generation:** Drug discovery
 - **Graph evolution:** Physical simulation

Examples of Node-Level ML Tasks

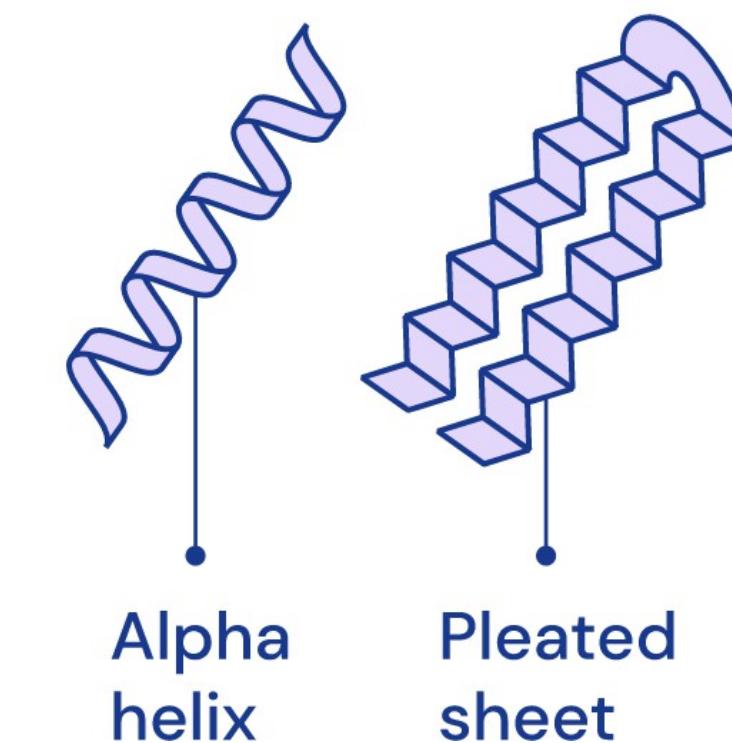
Example (1) Protein folding

A protein chain acquires its native 3D structure

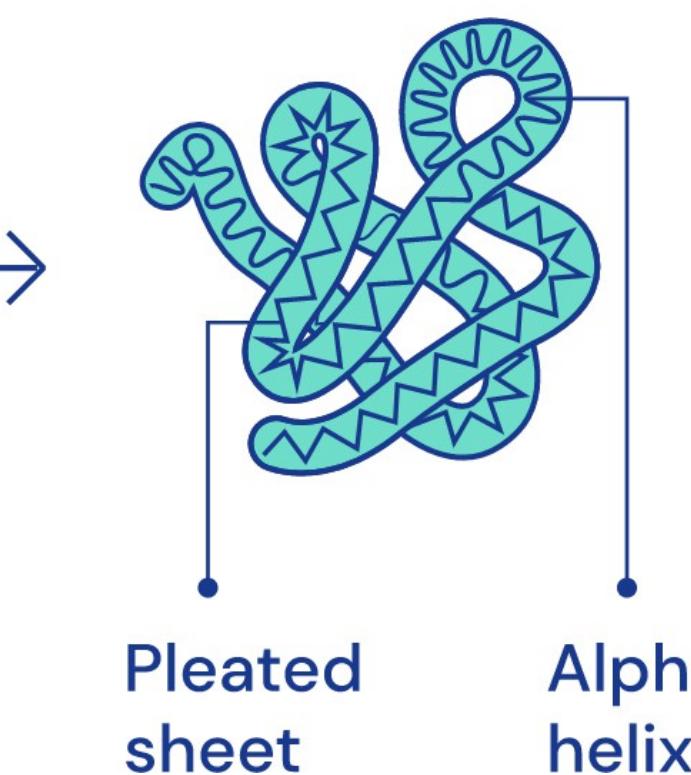
Every protein is made up of a sequence of amino acids bonded together



These amino acids interact locally to form shapes like helices and sheets



These shapes fold up on larger scales to form the full three-dimensional protein structure



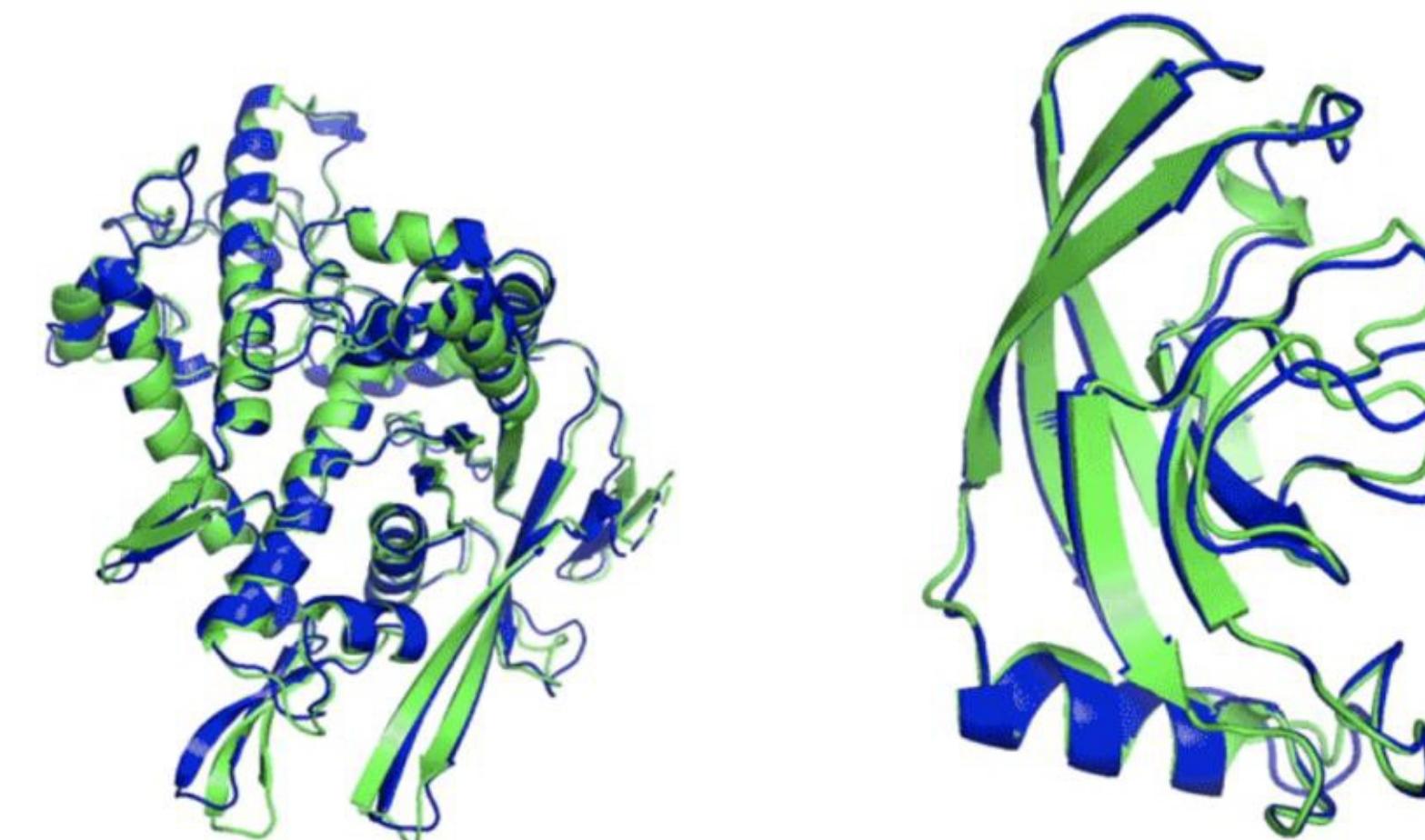
Proteins can interact with other proteins, performing functions such as signalling and transcribing DNA



Image credit: DeepMind

The Protein Folding Problem

Computationally predict a protein's 3D structure based solely on its amino acid sequence

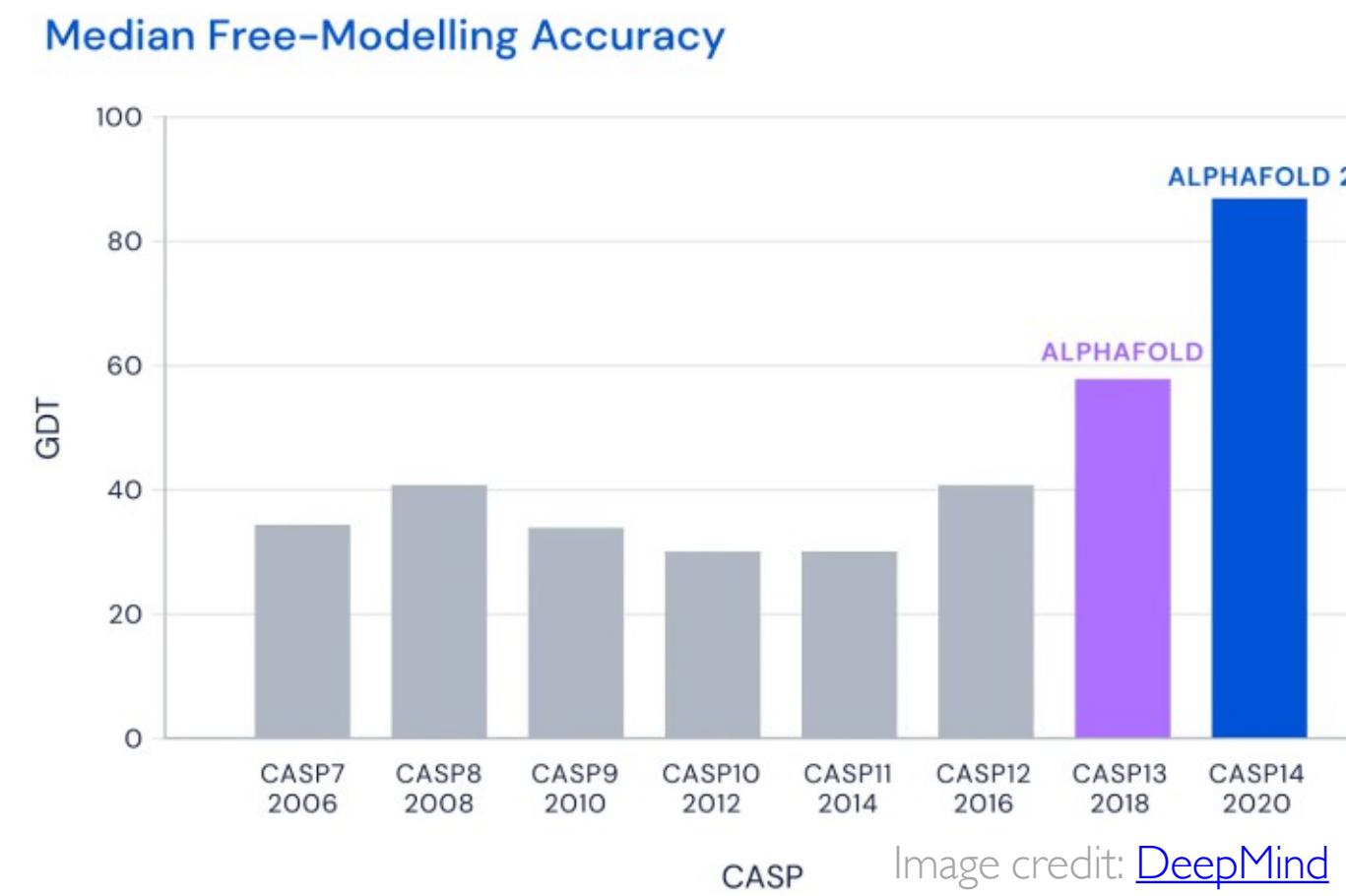


T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

T1049 / 6y4f
93.3 GDT
(adhesin tip)

- Experimental result
- Computational prediction

AlphaFold: Impact



AlphaFold's AI could change the world of biological science as we know it

DeepMind's latest AI breakthrough can accurately predict the way proteins fold

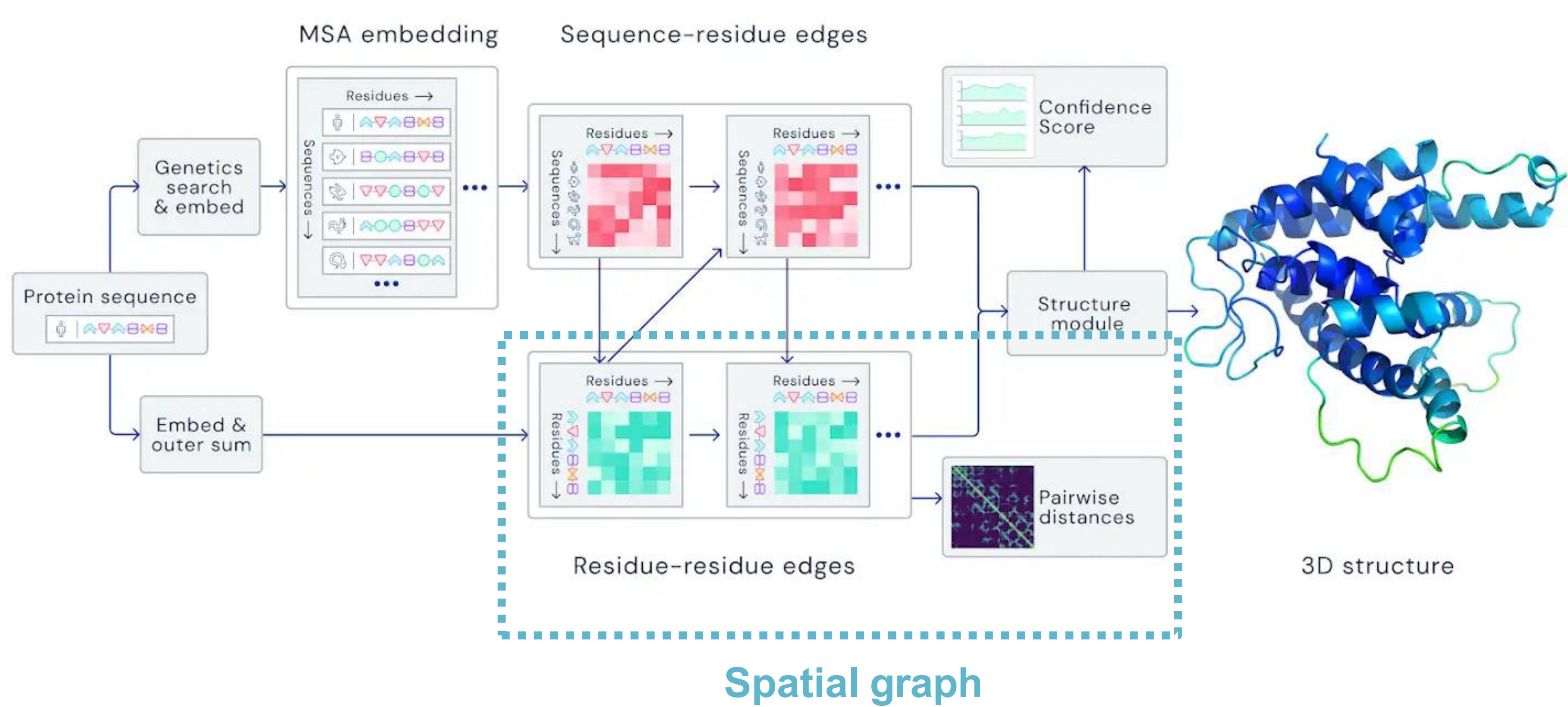
Has Artificial Intelligence 'Solved' Biology's Protein-Folding Problem?

12-14-20

DeepMind's latest AI breakthrough could turbocharge drug discovery

AlphaFold: Solving Protein Folding

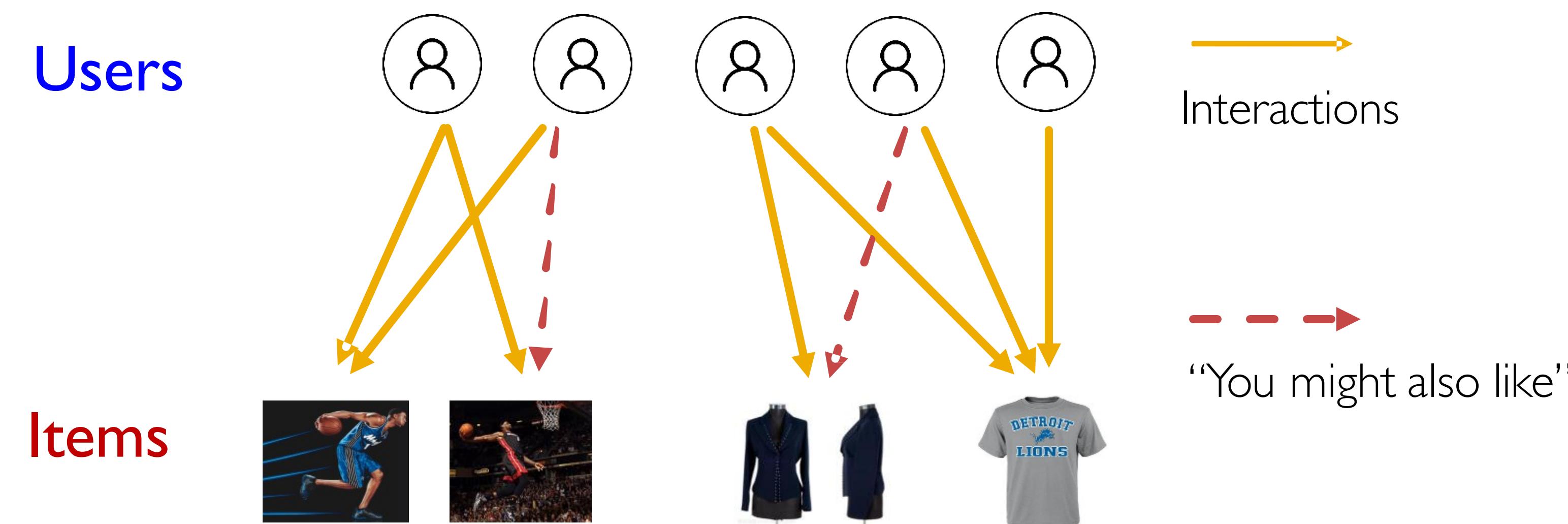
- Key idea: “Spatial graph”
 - Nodes: Amino acids in a protein sequence
 - Edges: Proximity between amino acids (residues)



Examples of Edge-Level ML Tasks

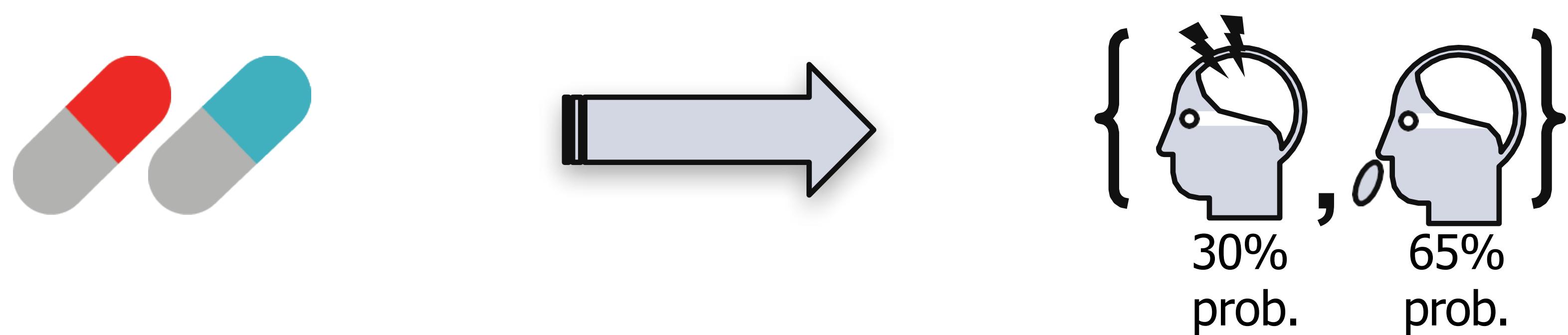
Example (2): Recommender Systems

- Users interacts with items
 - Watch movies, buy merchandise, listen to music
 - Nodes: Users and items
 - Edges: User-item interactions
- Goal: Recommend items users might like



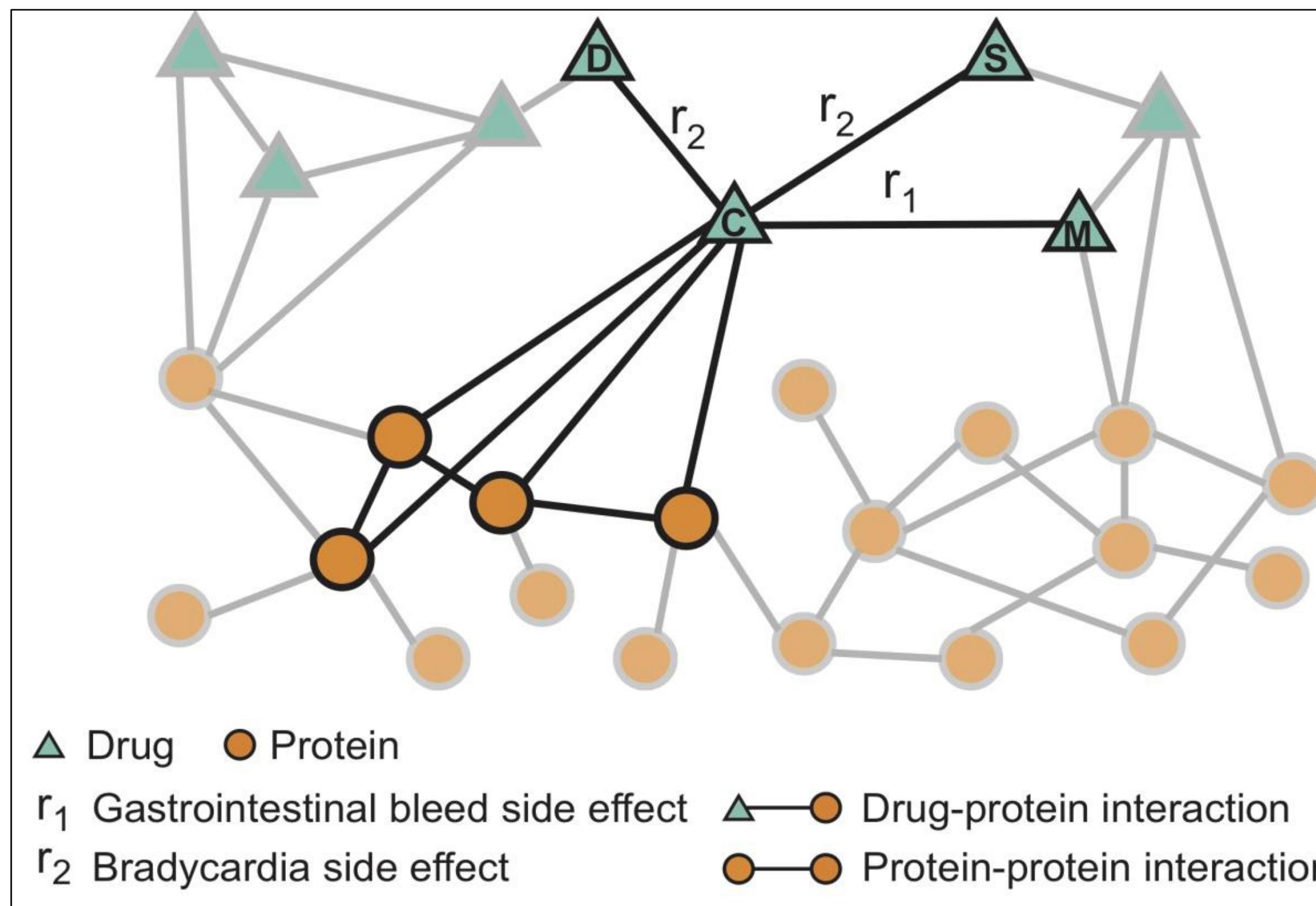
Example (3): Drug Side Effects

- Many patients take multiple drugs to treat complex or co-existing diseases:
- 46% of people ages 70-79 take more than 5 drugs
- Many patients take more than 20 drugs to treat heart disease, depression, insomnia, etc.
- Task: Given a pair of drugs predict adverse side effects



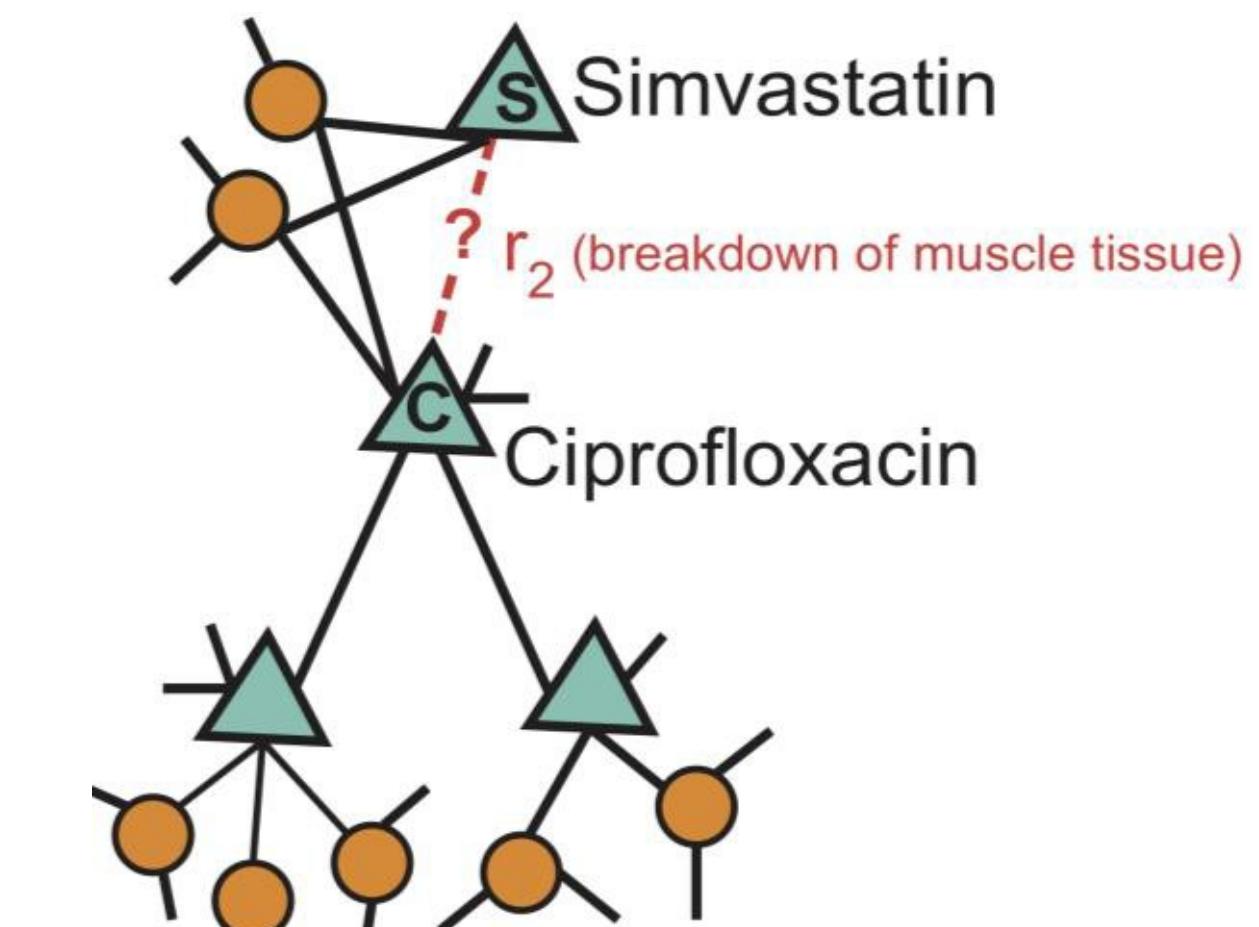
Biomedical Graph Link Prediction

- **Nodes:** Drugs & Proteins
- **Edges:** Interactions



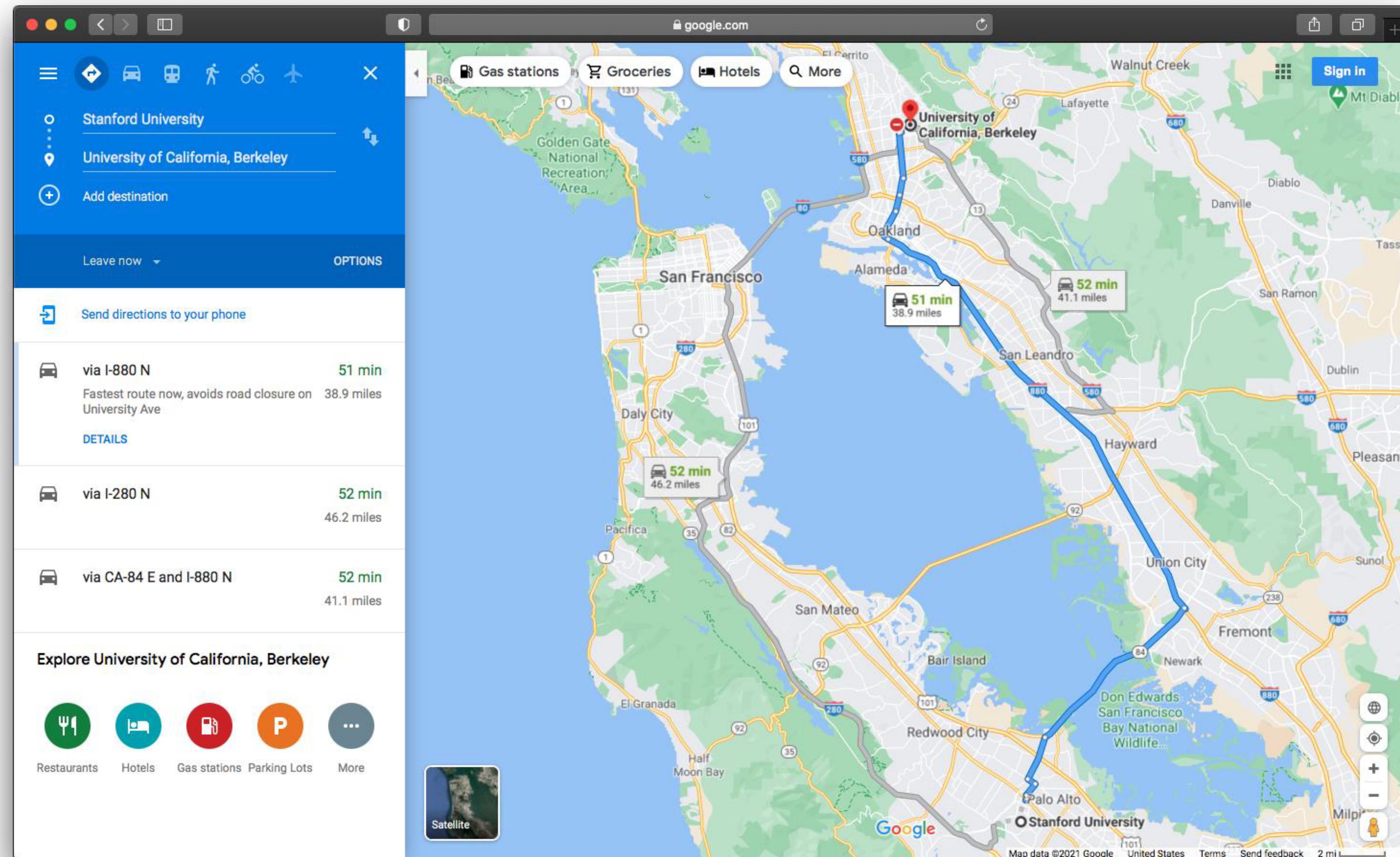
Query:

How likely will Simvastatin and Ciprofloxacin, when taken together, break down muscle tissue?



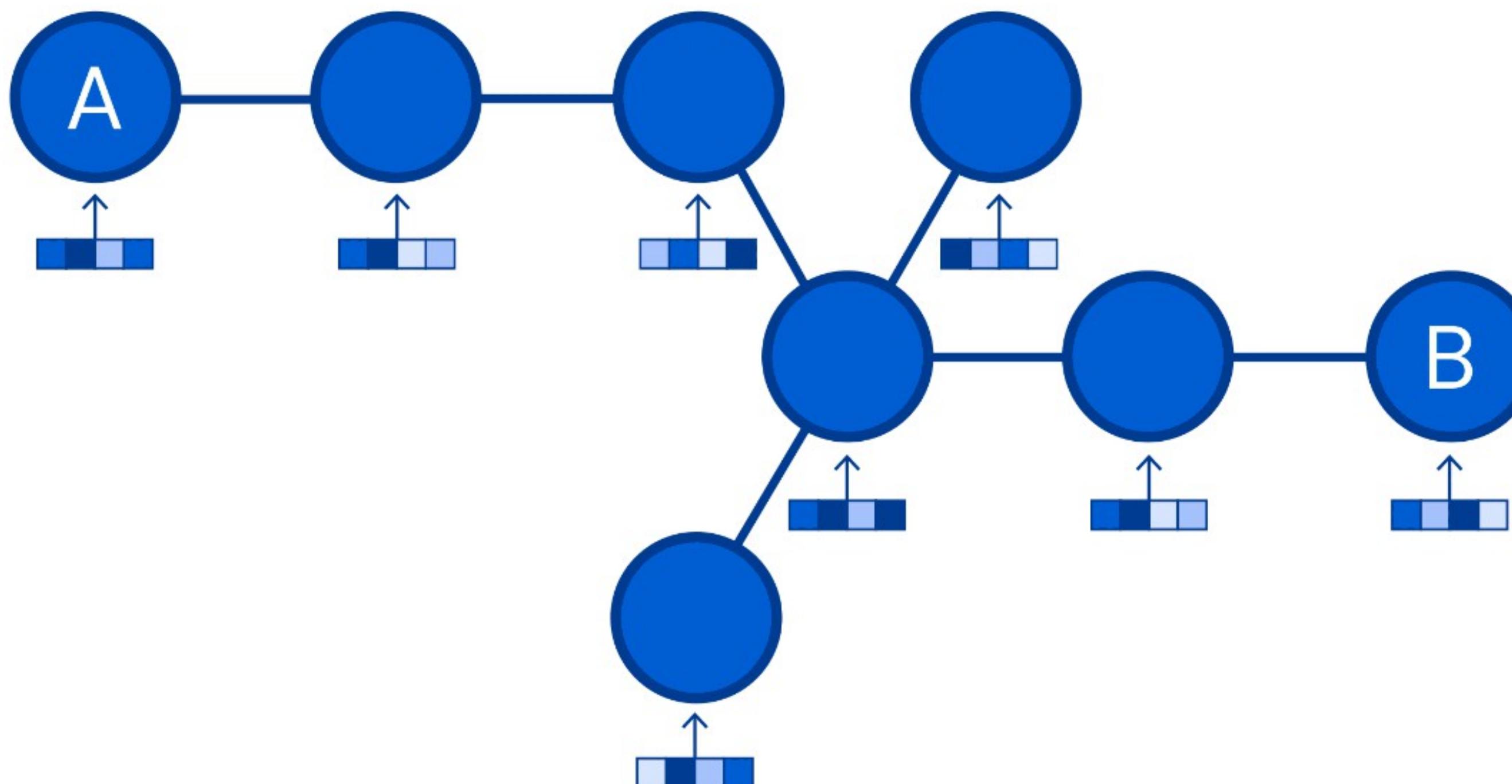
Examples of Subgraph-Level ML Tasks

Example (4): Traffic Prediction

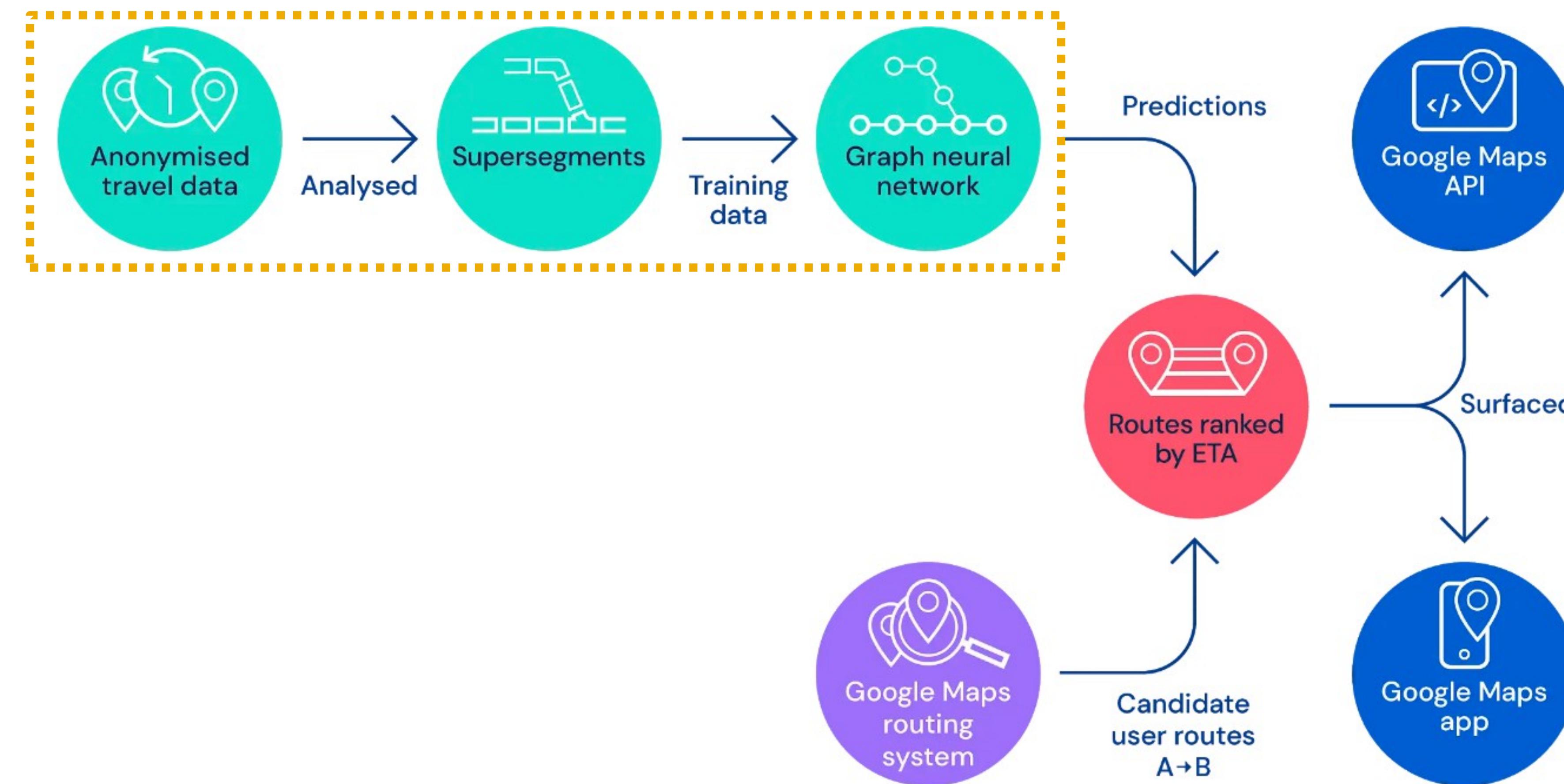


Road Network as a Graph

- Nodes: Road segments
- Edges: Connectivity between road segments
- Prediction: Time of Arrival (ETA)



Traffic Prediction via GNN



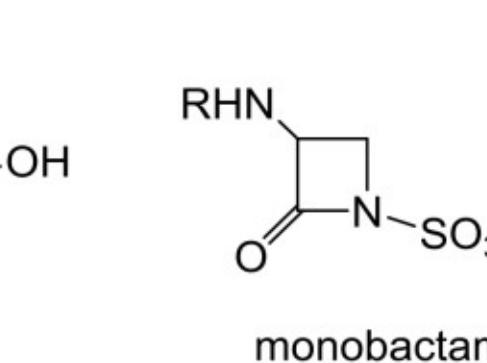
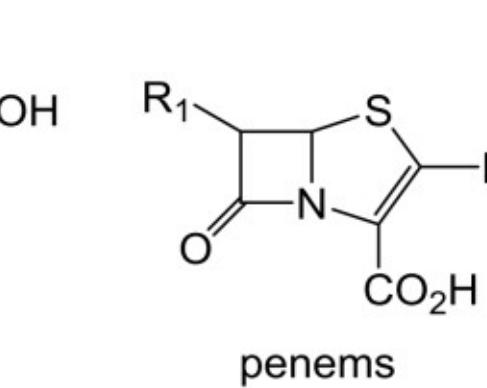
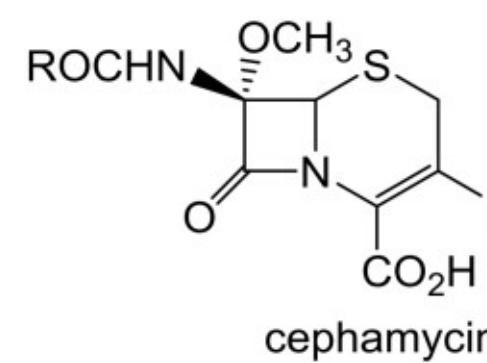
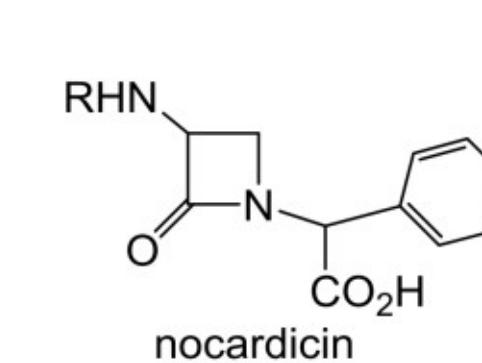
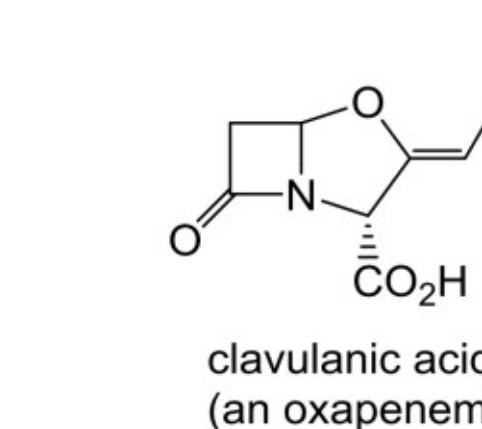
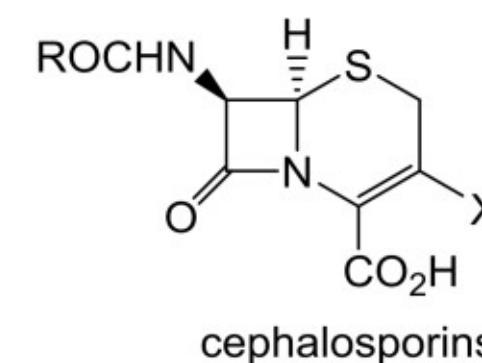
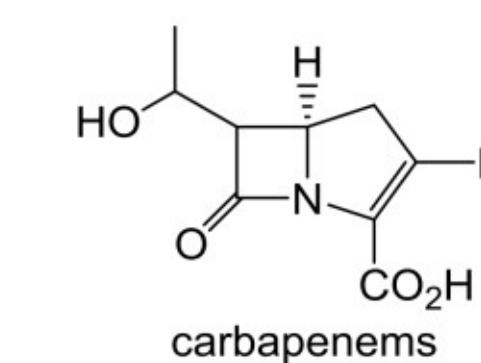
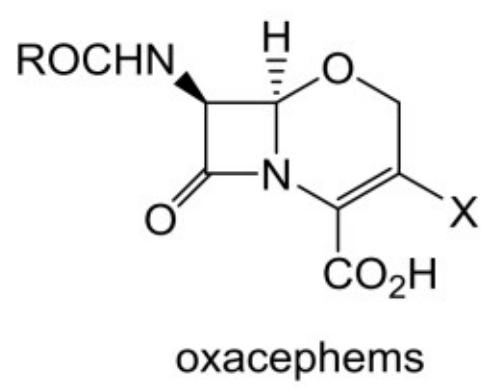
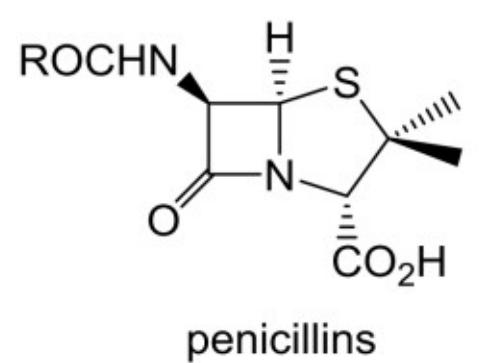
THE MODEL ARCHITECTURE FOR DETERMINING OPTIMAL ROUTES AND THEIR TRAVEL TIME.

Image credit: [DeepMind](#)

Examples of Graph-Level ML Tasks

Example (5): Drug Discovery

- Antibiotics are small molecular graphs
 - Nodes: Atoms
 - Edges: Chemical bonds

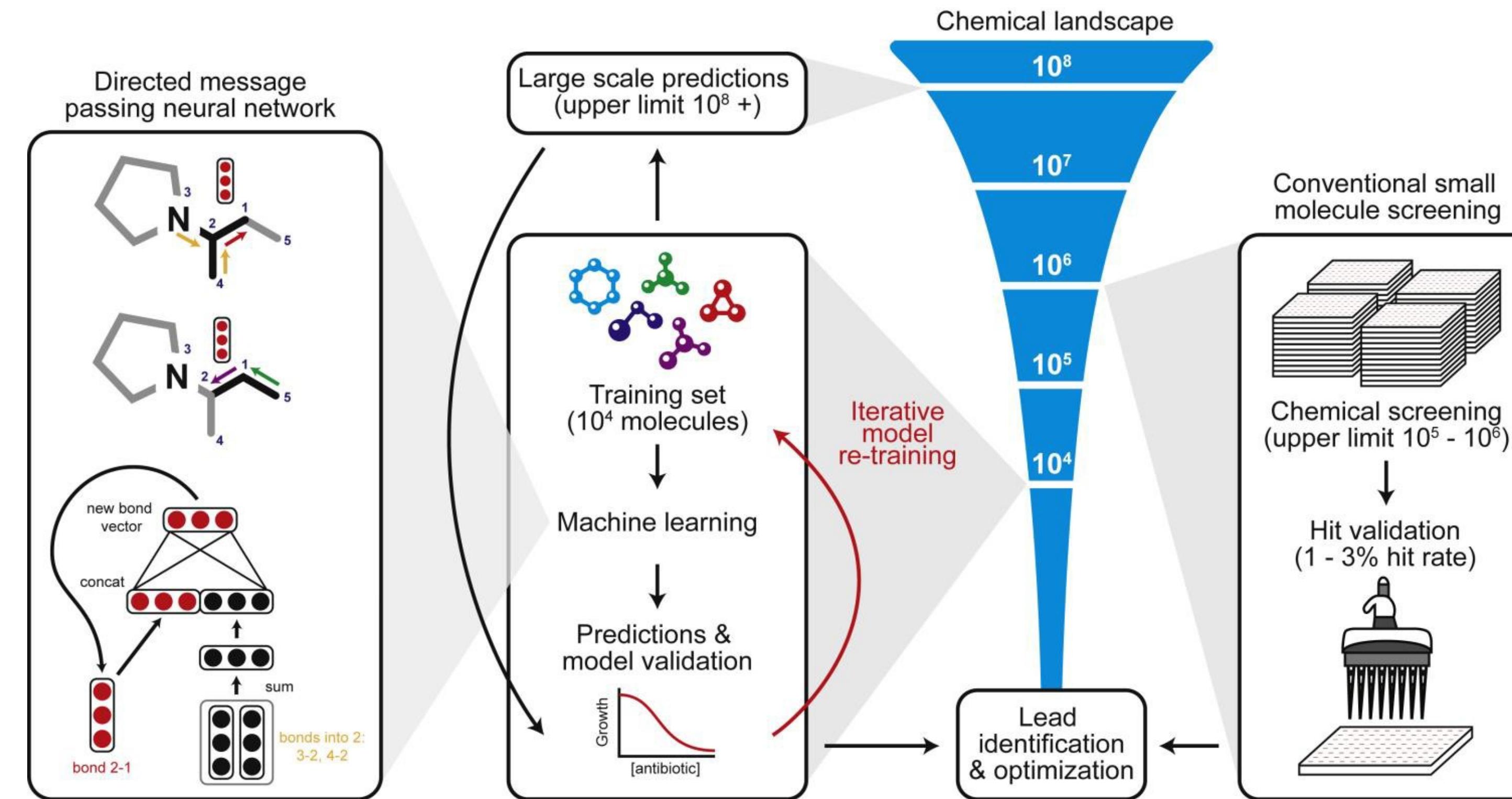


Konaklieva, Monika I. "Molecular targets of β-lactam-based antimicrobials: beyond the usual suspects." *Antibiotics* 3.2 (2014): 128-142.

Image credit: [CNN](#)

Deep Learning for Antibiotic Discovery

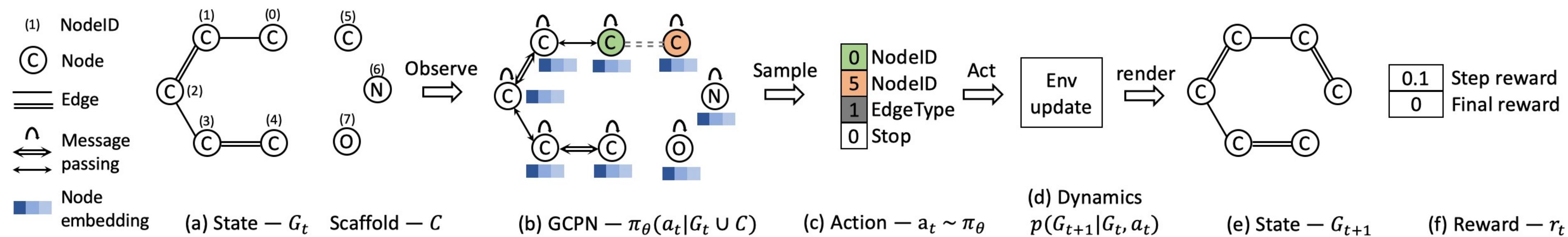
- A Graph Neural Network graph classification model
- Predict promising molecules from a pool of candidates



Stokes, Jonathan M., et al. "A deep learning approach to antibiotic discovery." Cell 180.4 (2020): 688-702.

Molecule Generation/Optimization

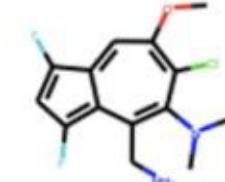
Graph generation: Generating novel molecules



Use case 1: Generate novel molecules with high Drug likeness value



0.948

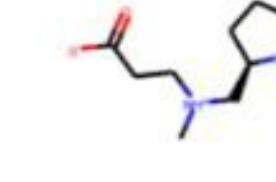


0.945

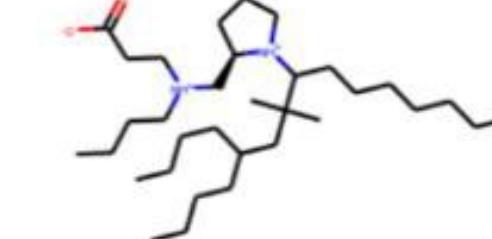


0.944

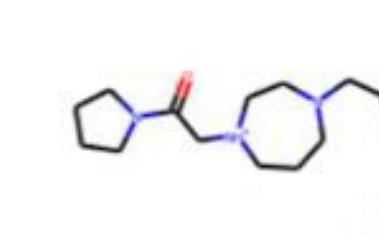
Use case 2: Optimize existing molecules to have desirable properties



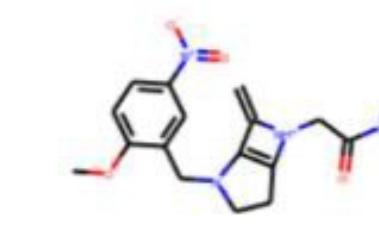
-8.32



-0.71



-5.55

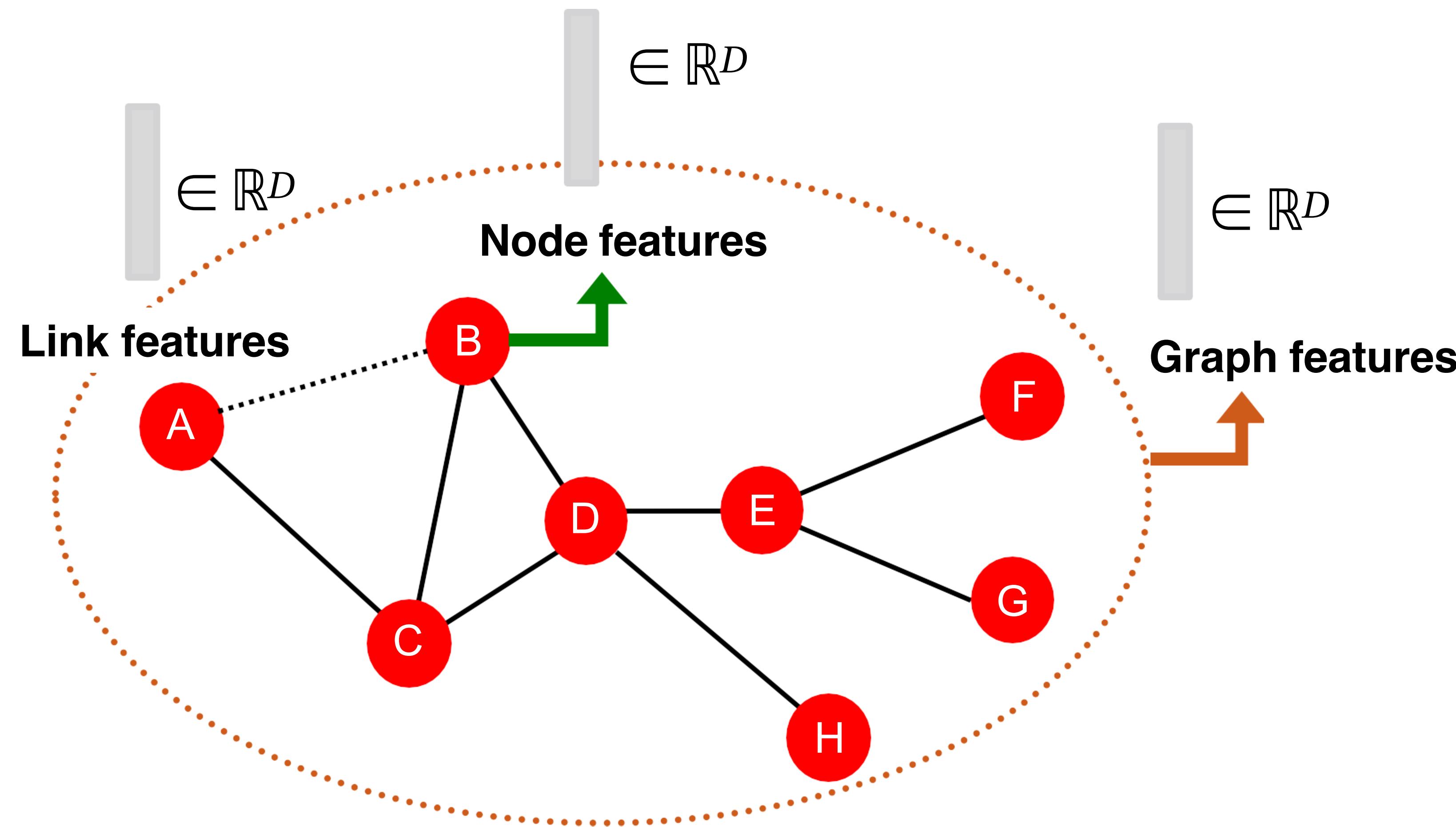


-1.78

Drug likeness

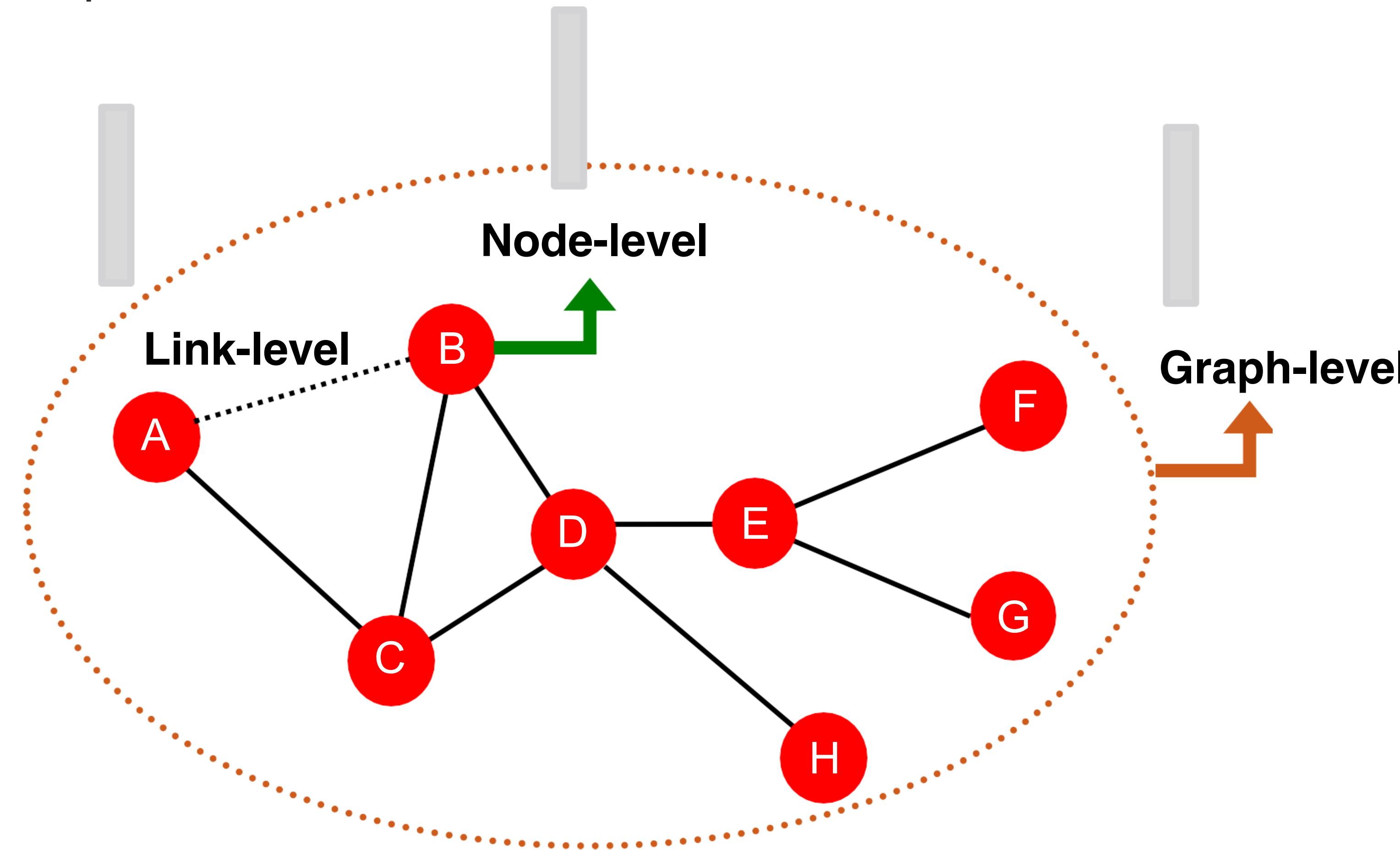
Traditional LM Pipeline

- Design features for nodes/links/graphs
- Obtain features for all training data



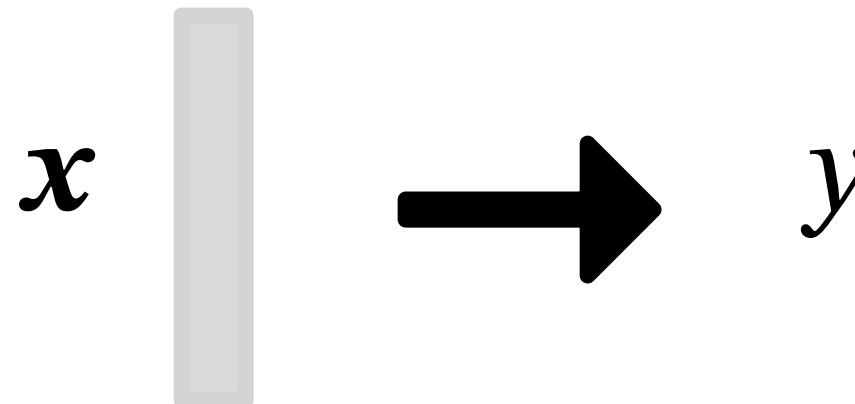
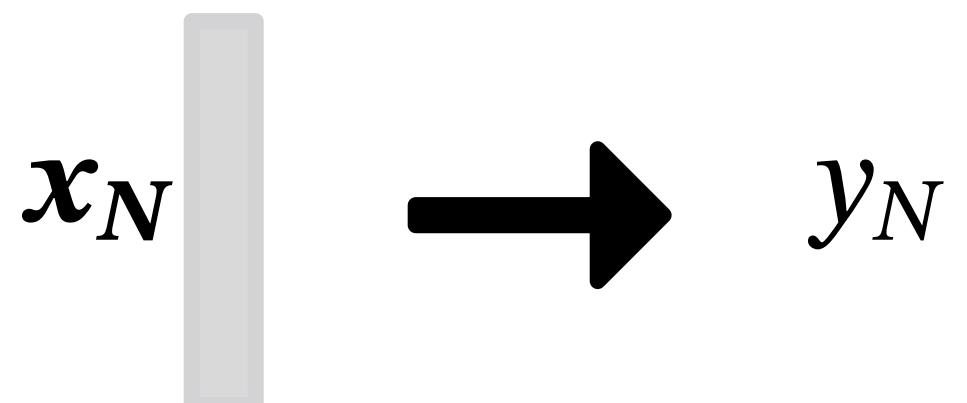
Machine Learning Tasks: Review

- **Node-level** prediction
- **Link-level** prediction
- **Graph-level** prediction



Traditional LM Pipeline

- **Train an ML model:**
 - Logistic Regression
 - Random forest
 - Neural network, etc.
- **Apply the model:**
 - Given a new node/link/graph
 - Obtain its features
 - Make a prediction



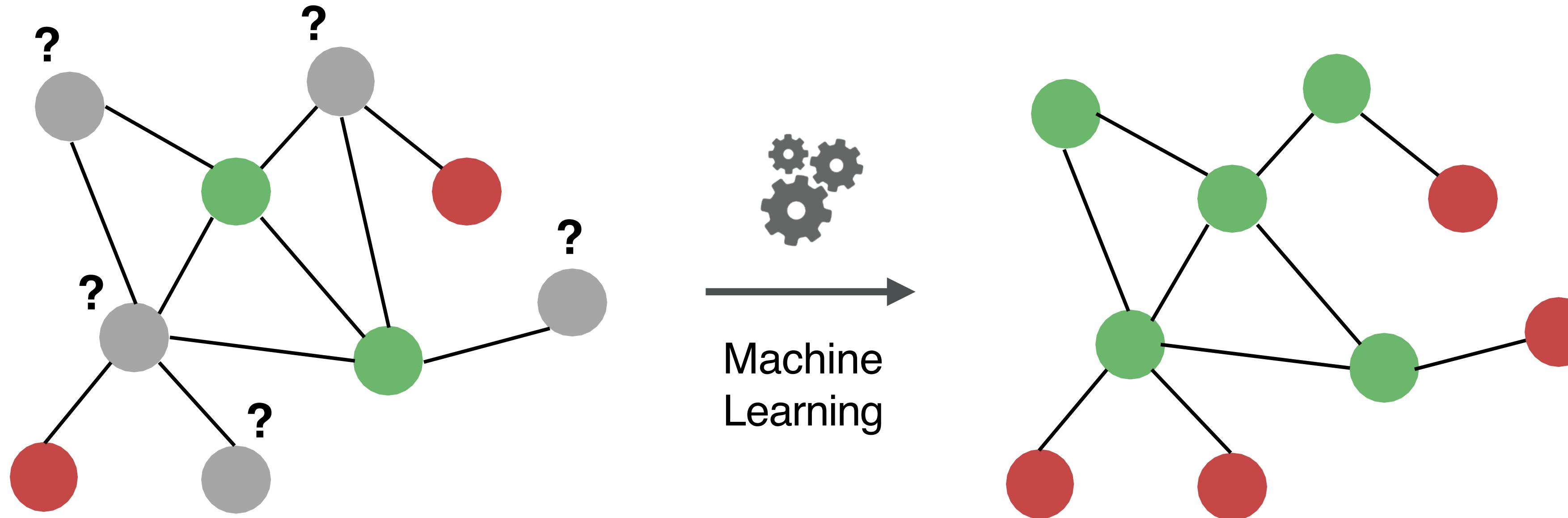
Features Design

- Using **effective features** x over graphs is the key to achieving good model performance.
- Traditional ML pipeline uses **hand-designed features**.
- In this lecture, we overview the traditional features for:
 - **Node-level** prediction
 - **Link-level** prediction
 - **Graph-level** prediction
- For simplicity, we focus on **undirected** graphs.

Machine Learning in Graphs

- **Goal:** Make predictions for a set of objects
- Design choices:
 - **Features:** d-dimensional vectors x
 - **Objects:** Nodes, edges, sets of nodes, entire graphs
 - **Objective function:**
 - What task are we aiming to solve?

Node-Level Tasks



Node classification

ML needs features.

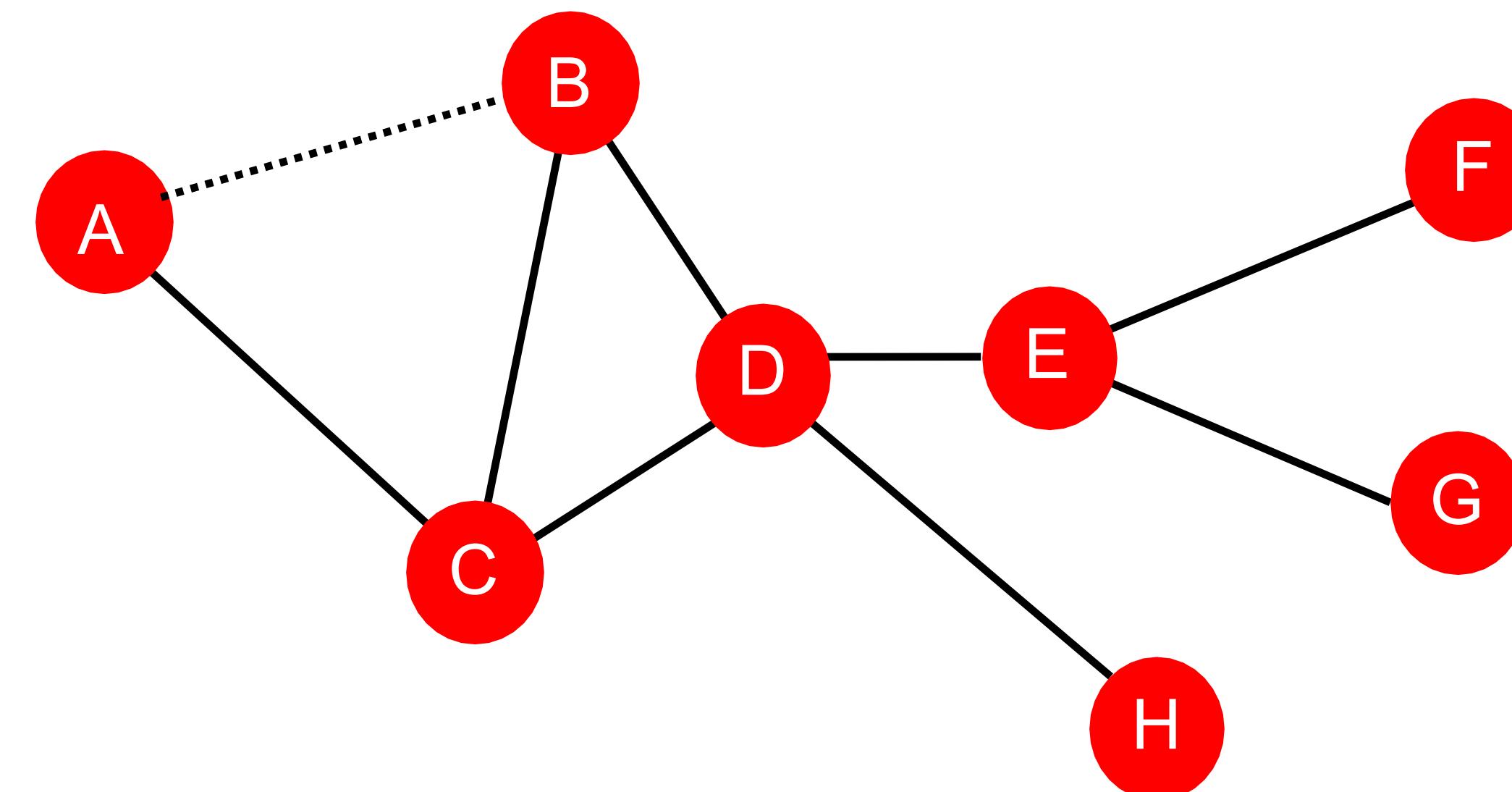
Node-Level Overview

Goal

Characterize the structure and position of a node in the network:

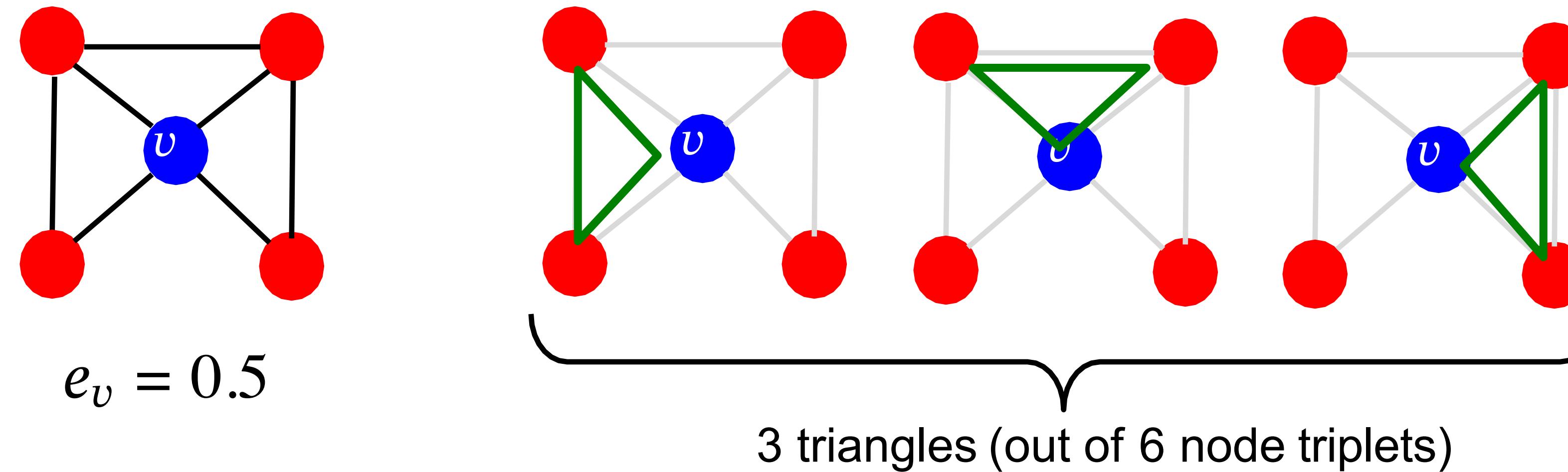
- Node degree
- Node centrality
- Clustering coefficient
- Graphlets

Node features



Node Features: Graphlets

- **Observation:** Clustering coefficient counts the **#triangles** in the ego network.



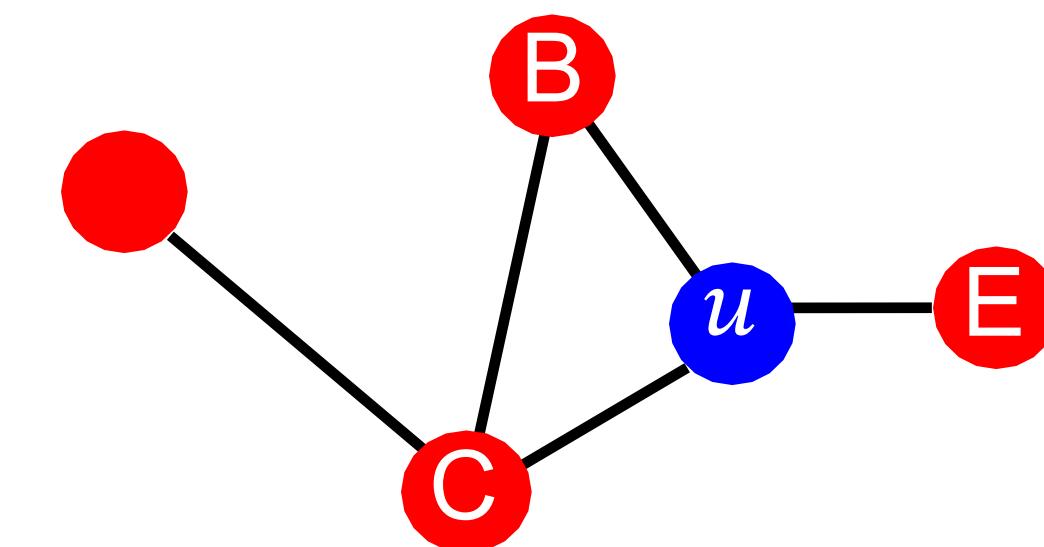
- We can generalize the above by counting **#(pre-specified subgraphs, i.e., graphlets)**

Node Features: Graphlets

- **Goal:** Describe the network structure around node u
 - **Graphlets are small subgraphs that describe the structure of node u 's network neighborhood.**

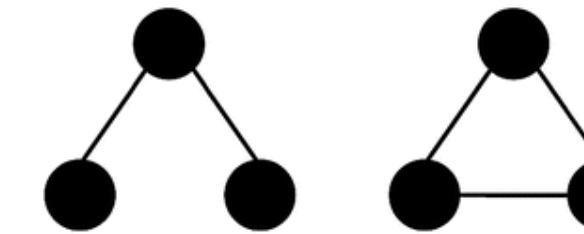
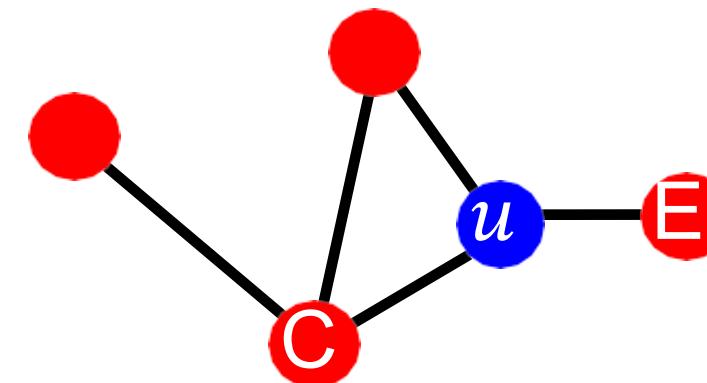
Analogy:

- Degree counts **#(edges)** that a node touches
- Clustering coefficient counts **#(of triangles)** that a node touches.
- **Graphlet Degree Vector (GDV):** Graphlet-base features for nodes
 - GDV counts **#(graphlets)** that a node touches



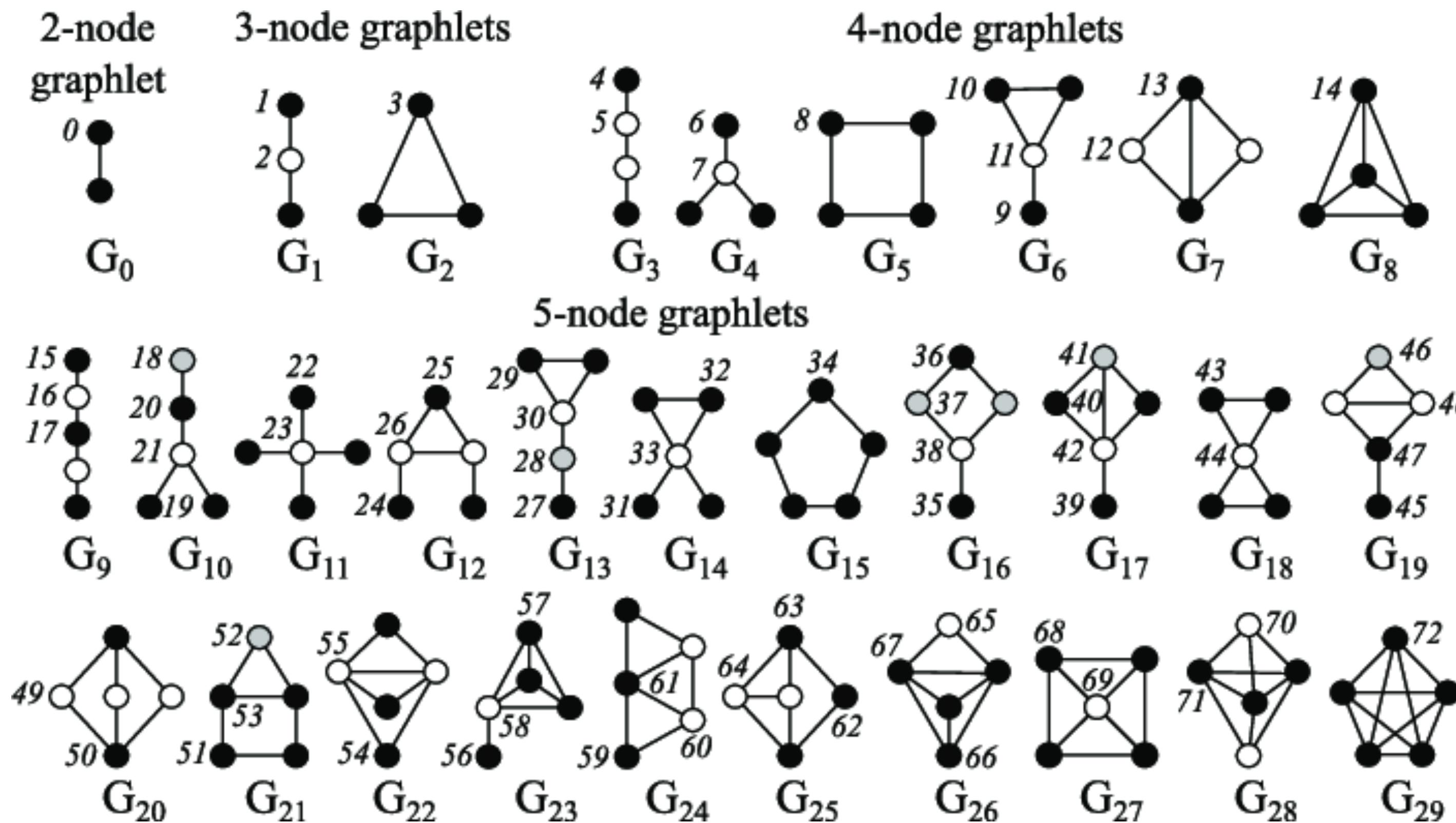
Node Features: Graphlets

- Considering graphlets of size 2-5 nodes we get:
 - A vector of 73 coordinates is a signature of a node that describes the topology of the node's neighborhood
 - Graphlet degree vector provides a measure of a **node's local network topology**:
 - Comparing vectors of two nodes provides a more detailed measure of local topological similarity than node degrees or clustering coefficient.



Node Features: Graphlets

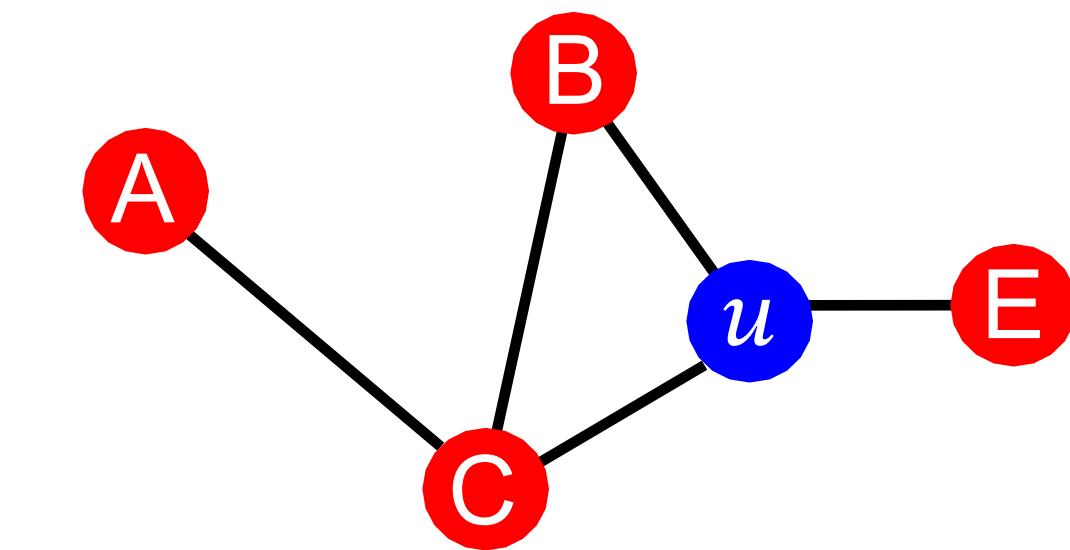
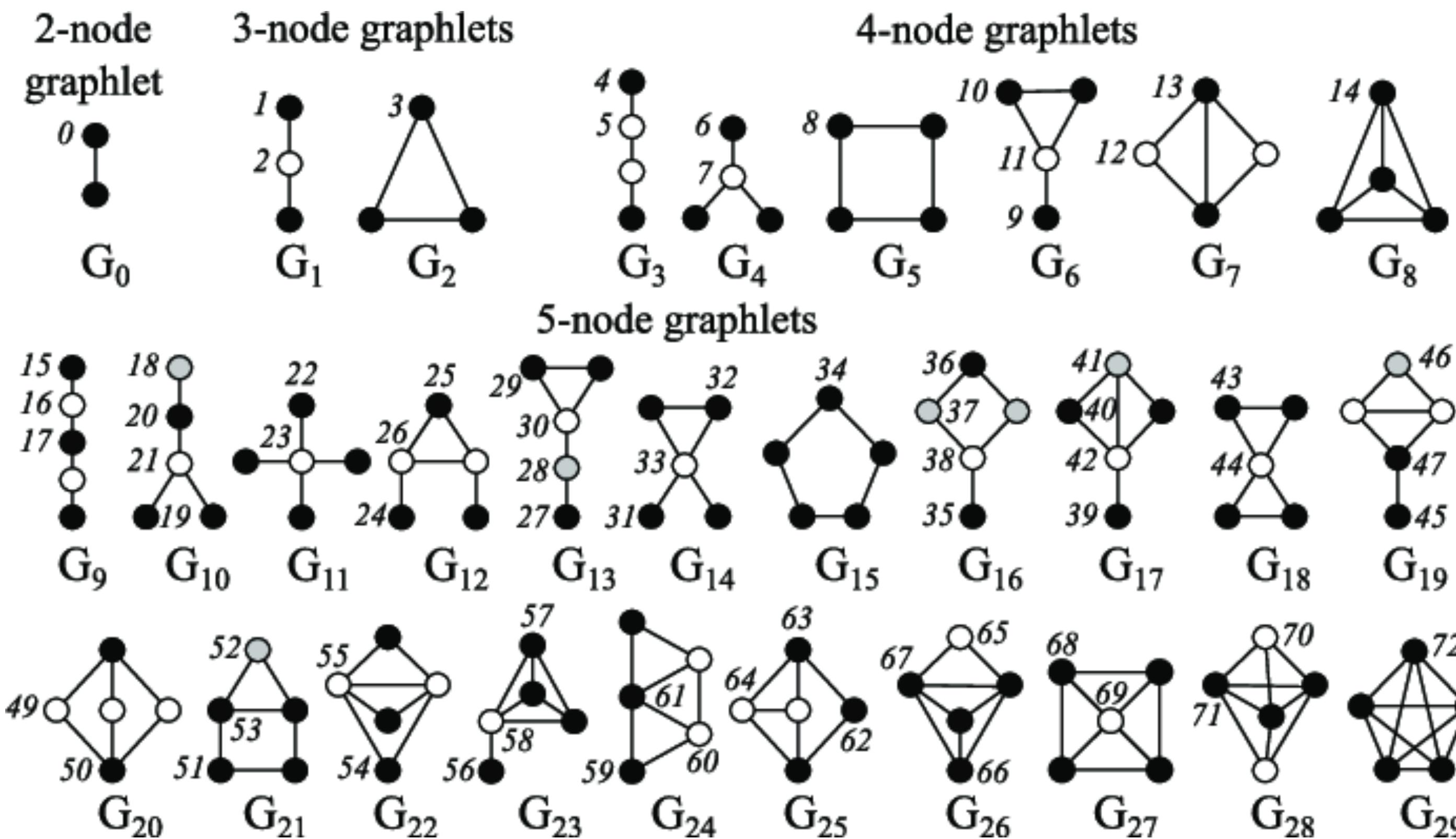
Rooted connected induced non-isomorphic subgraphs



There are 73 different graphlets on up to 5 nodes

Node Features: Graphlets

Rooted connected induced non-isomorphic subgraphs



u has graphlets:
0, 1, 2, 3, 5, 10, 11, ...

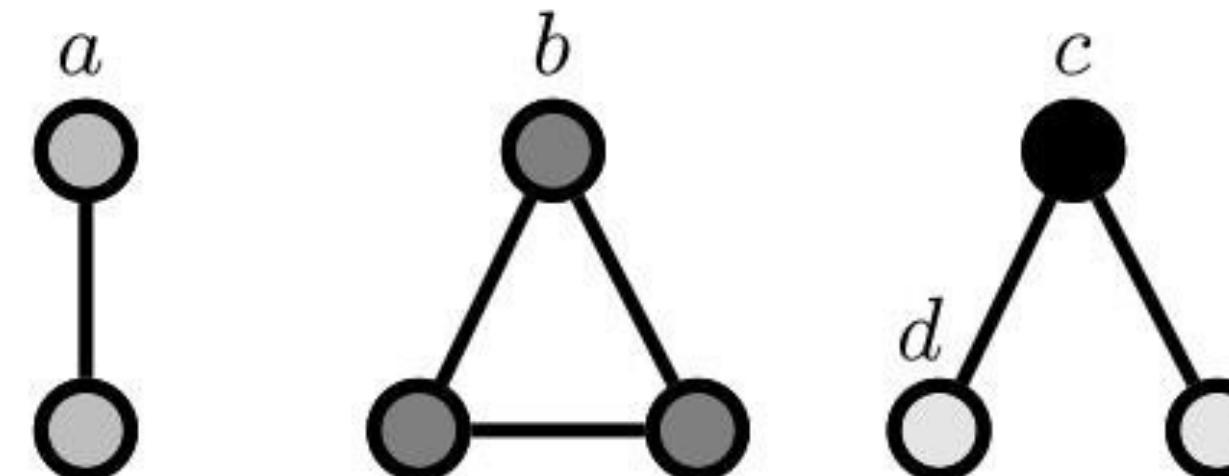
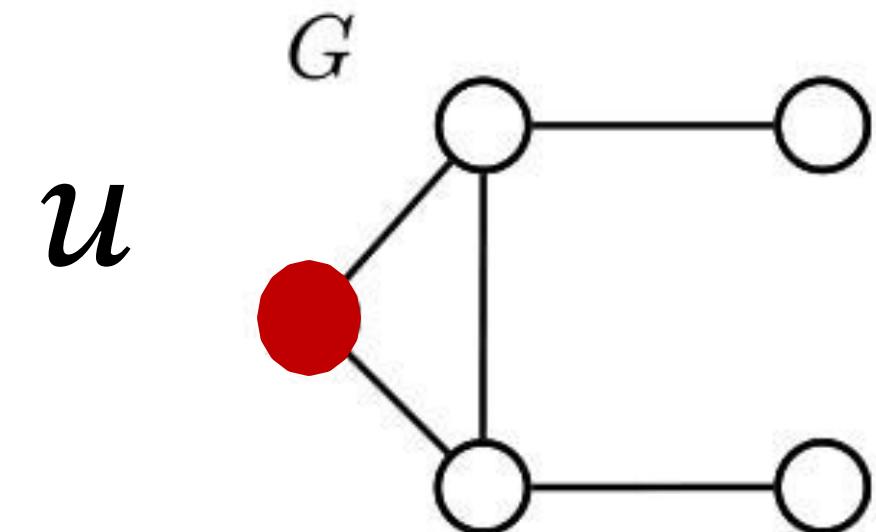
There are 73 different graphlets on up to 5 nodes

Node Features: Graphlets

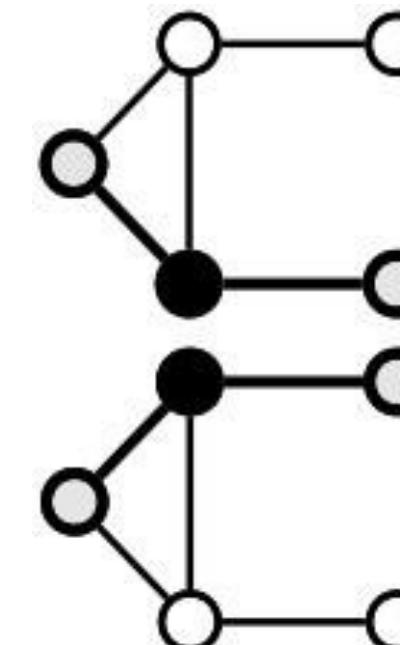
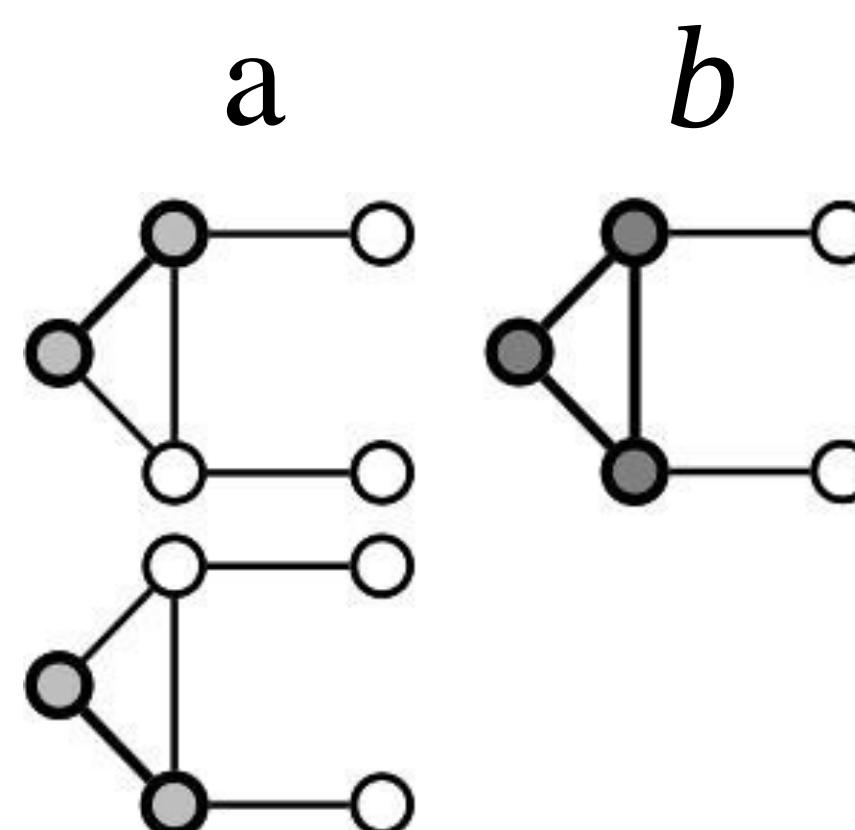
- **Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.

- **Example:**

Possible graphlets on up to 3 nodes



Graphlet instances of node u :



GDV of node u :

a, b, c, d

$[2, 1, 0, 2]$

Node-Level Features: Summary

- We have introduced different ways to obtain node features.
- They can be categorized as:
 - Importance-based features:
 - Node degree
 - Different node centrality measures
 - Structure-based features:
 - Node degree
 - Clustering coefficient
 - Graphlet count vector

Node-Level Features: Summary

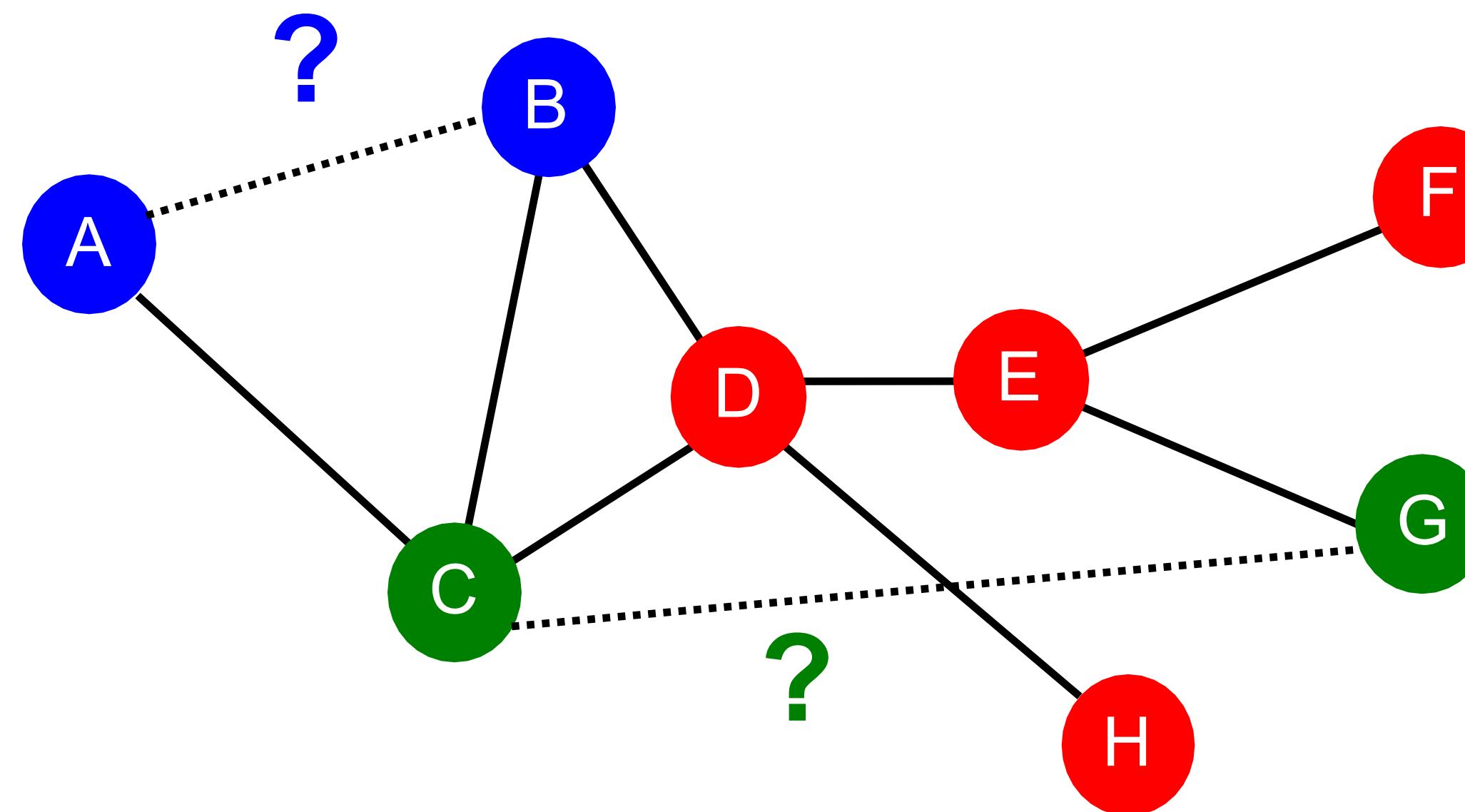
- **Importance-based** features: capture the **importance** of a node in a graph
 - **Node degree:**
 - Counts the number of neighboring nodes
 - **Node centrality:**
 - Models the importance of neighboring nodes in a graph
 - Different modeling choices: eigenvector centrality, betweenness centrality, closeness centrality
- Useful for predicting influential nodes in a graph
 - Example: predicting celebrity users in a social network

Node-Level Features: Summary

- **Structure-based** features: capture **topological properties** of the local neighborhood around a node.
 - **Node degree:**
 - Counts the number of neighboring nodes
 - **Clustering coefficient:**
 - Measures how connected neighboring nodes are
 - **Graphlet degree vector:**
 - Counts the occurrences of different graphlets
- Useful for predicting a particular role a node plays in a graph:
 - Example: predicting protein functionality in a protein-protein interaction network.

Link-Level Prediction Task: Recap

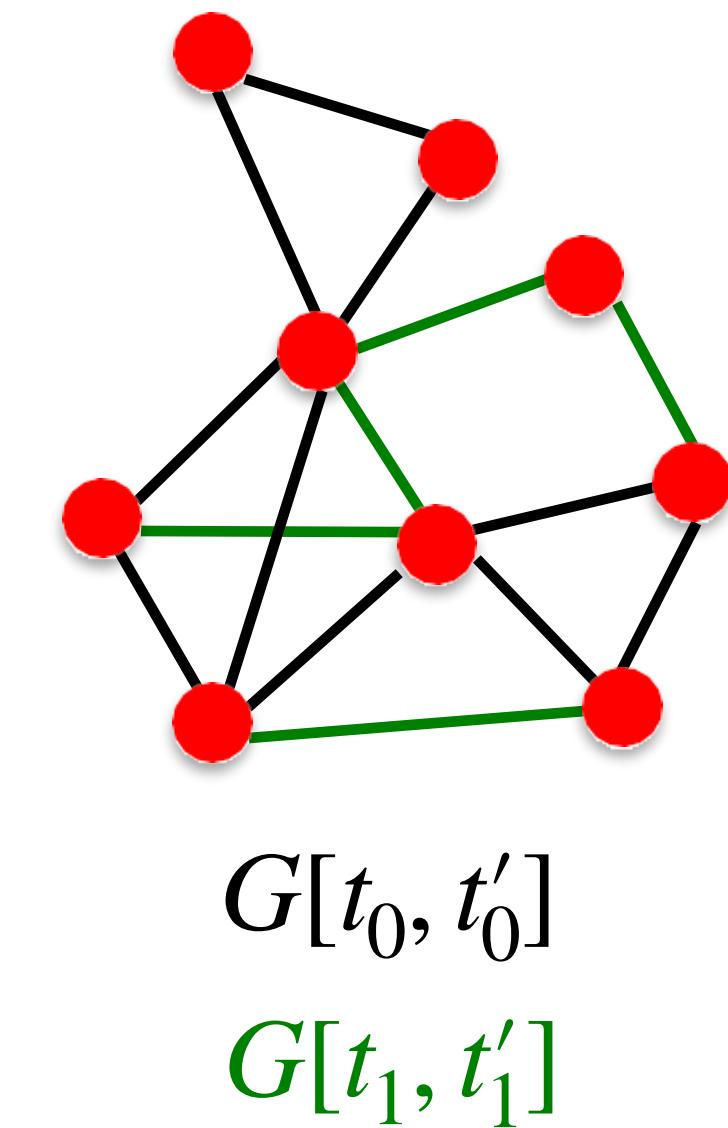
- The task is to predict new links based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for a pair of nodes.



Link Prediction as a Task

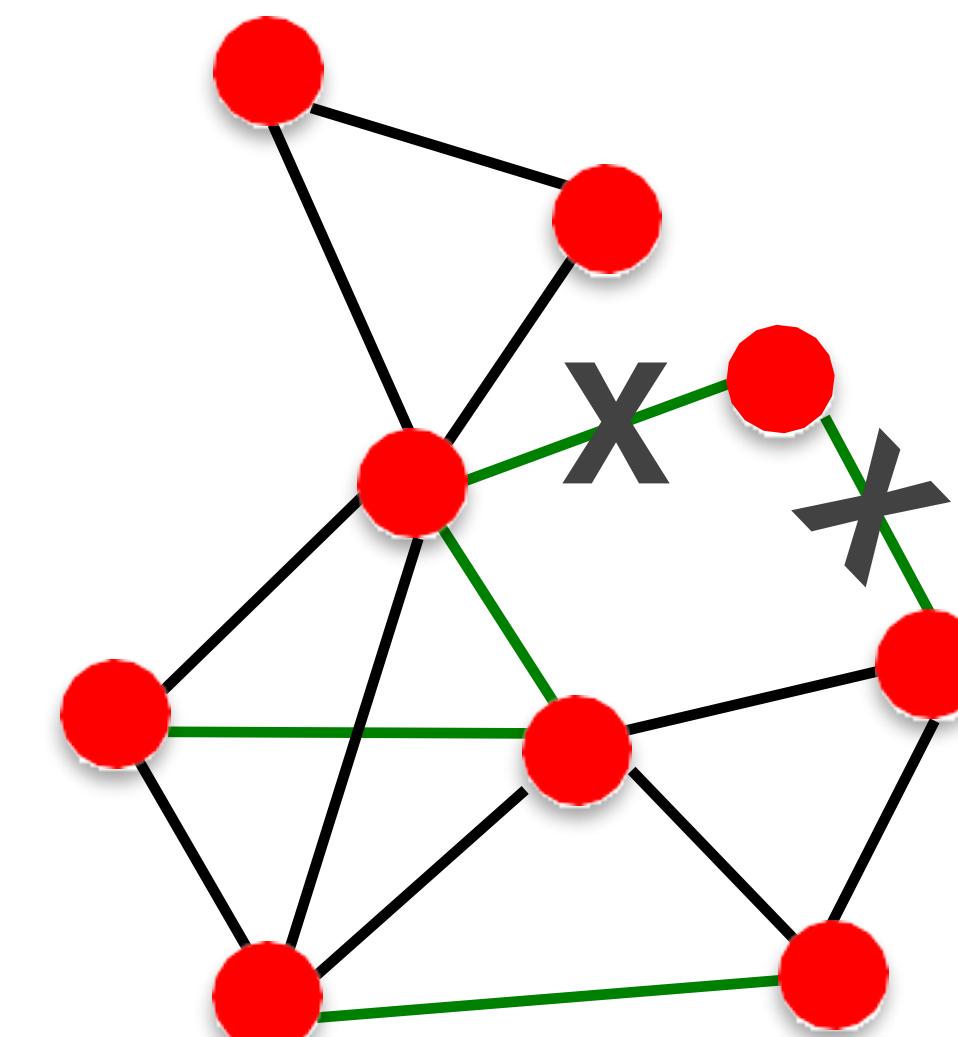
Two formulations of the link prediction task:

- **Links missing at random**:
 - Remove a random set of links and then aim to predict them
- **Links over time**:
 - Given $G[t_0, t'_0]$ a graph defined by edges up to time t'_0 , output a ranked list L of edges (not in $G[t_0, t'_0]$) that are predicted to appear in time $G[t_1, t'_1]$
- Evaluation:
 - $n = |E_{new}|$: # new edges that appear during the test period $G[t_1, t'_1]$
 - Take the top n elements of L and count the correct edges



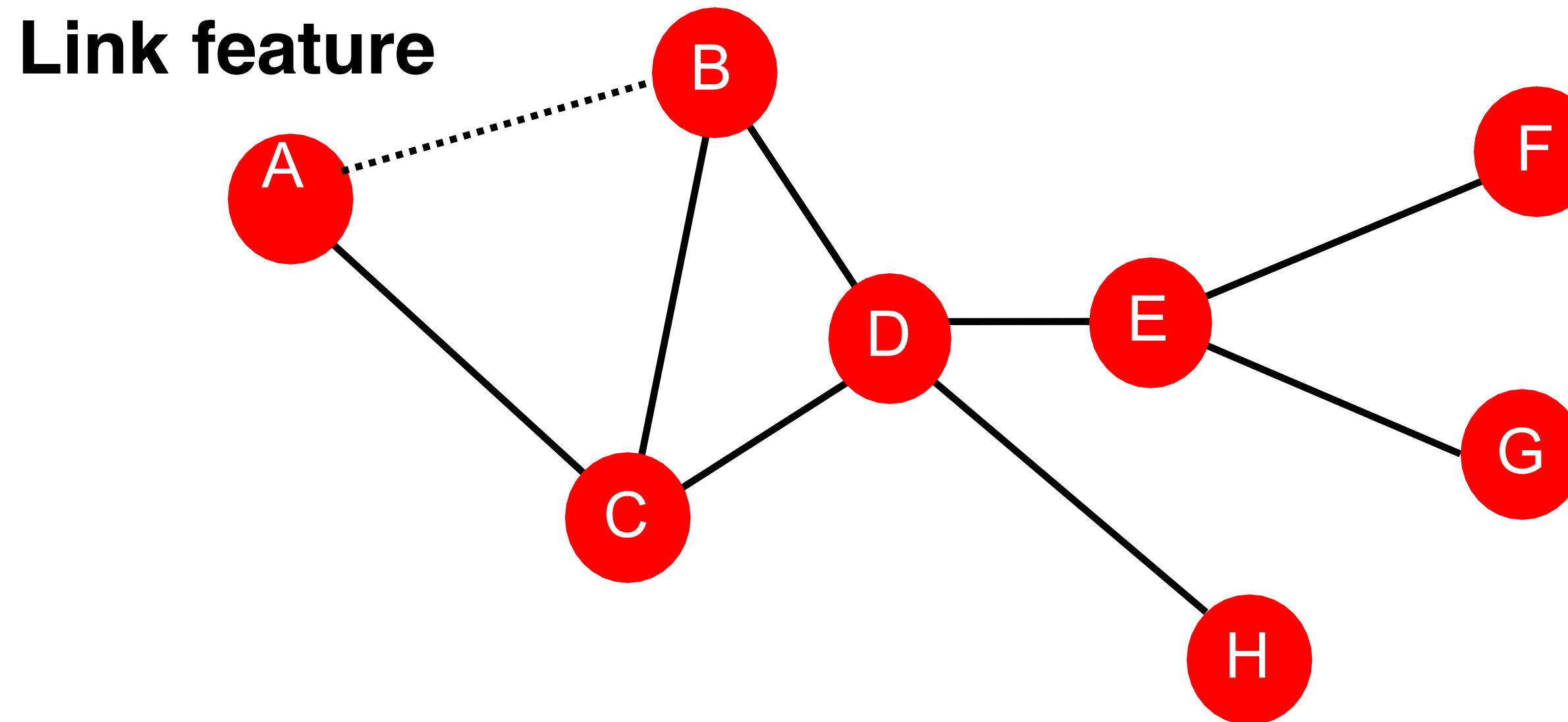
Link Prediction via Proximity

- **Methodology:**
 - For each pair of nodes (x,y) compute score $c(x,y)$
 - For example, $c(x,y)$ could be the # of common neighbors of x and y
 - Sort pairs (x,y) by the decreasing score $c(x,y)$
 - Predict top n pairs as new links
 - See which of these links appear in $G[t_1, t'_1]$



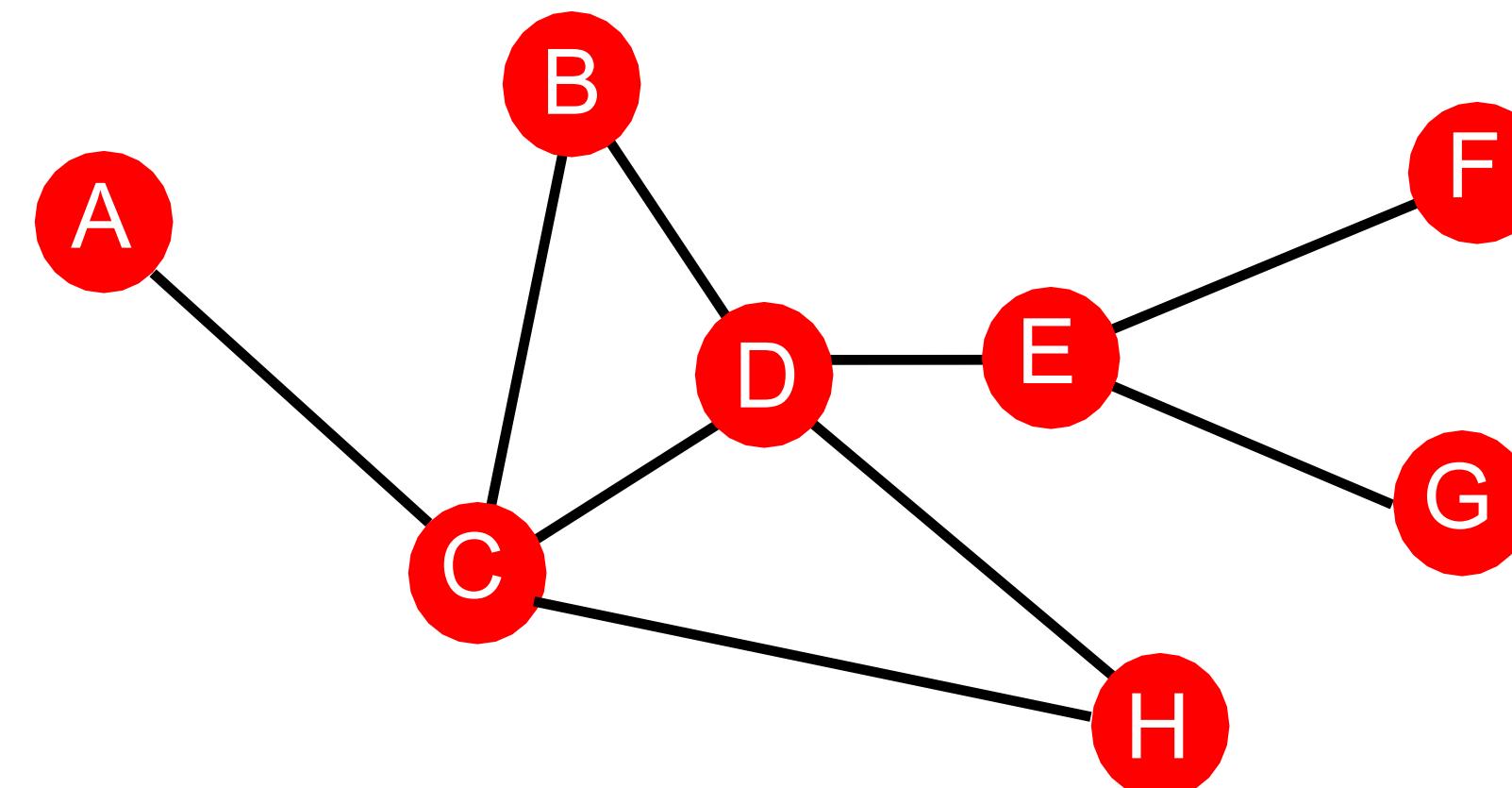
Link-Level Features: Overview

- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



Distance-based Features

- **Shortest-path distance between two nodes**
- However, this **does not capture the degree of neighborhood overlap**:
 - Node pair (B, H) has 2 shared neighboring nodes, while pairs (B, E) and (A, B) only have 1 such node.
- Example:



$$S_{BH} = S_{BE} = S_{AB} = 2$$

$$S_{BG} = S_{BF} = 3$$

Local Neighborhood Overlap

Captures # neighboring nodes shared between two nodes v_1 and v_2

- **Common neighbors** $|N(v_1) \cap N(v_2)|$

- **Example:**

$$|N(A) \cap N(B)| = |\{C\}| = 1$$

- **Jaccard's coefficient**

$$\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$$

- **Example:**

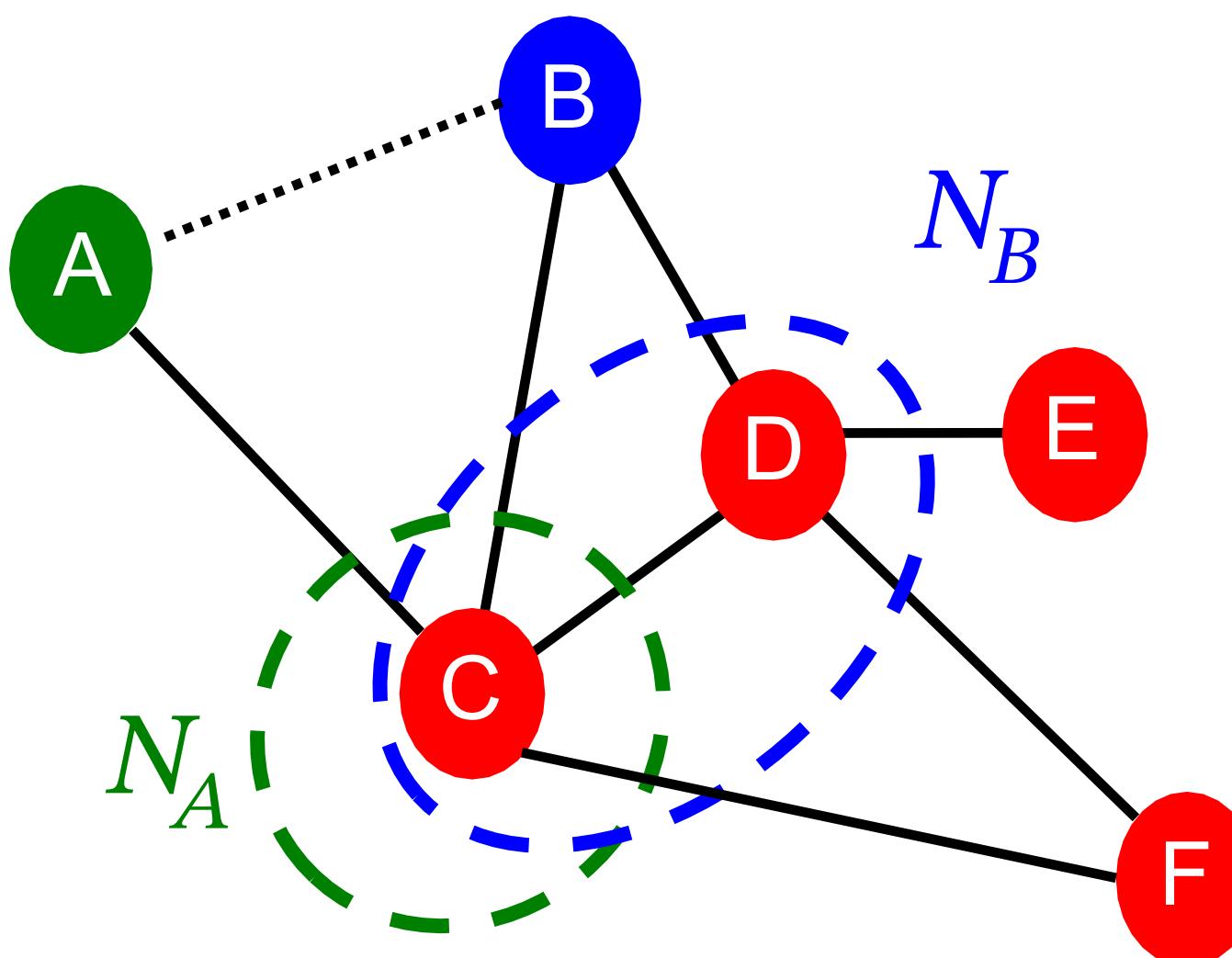
$$\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$$

- **Adamic-Adar index**

- **Example:**

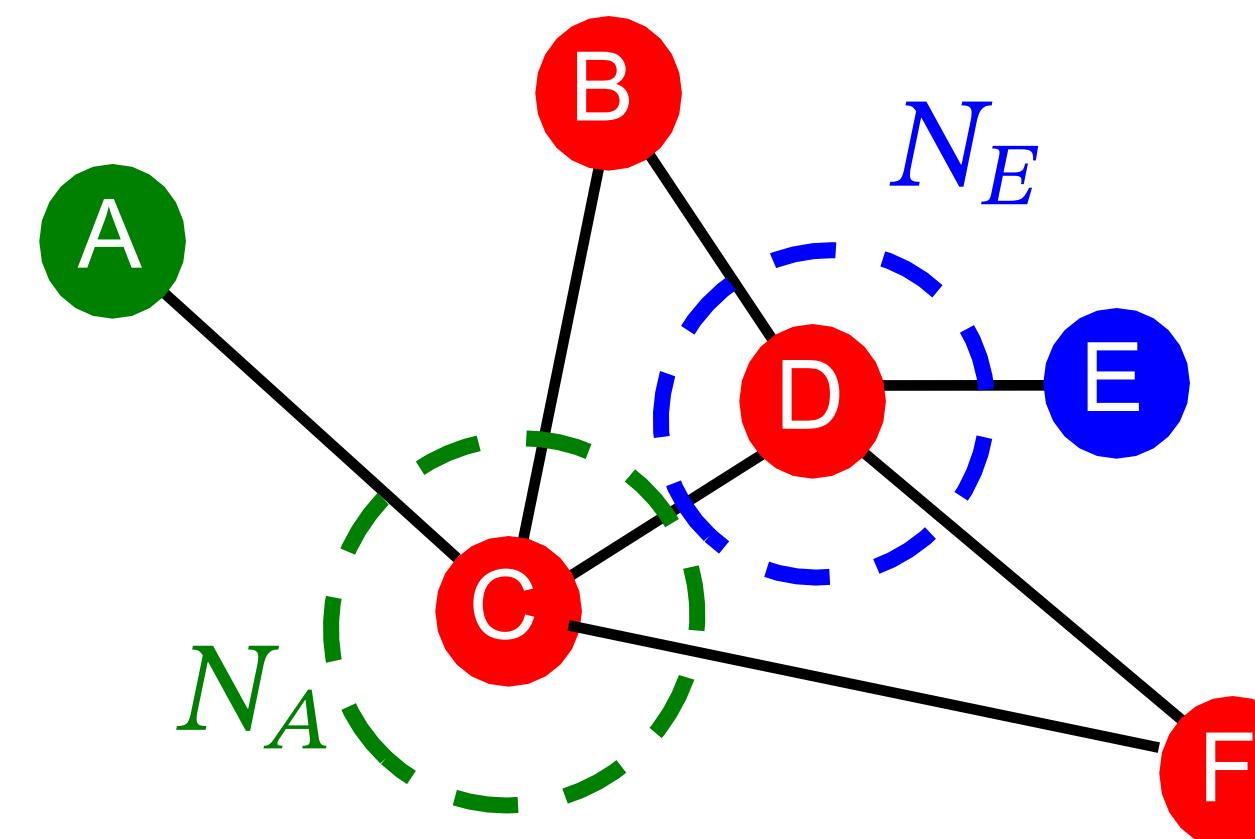
$$\frac{1}{\log(k_C)} = \frac{1}{\log 4}$$

$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$



Global Neighborhood Overlap

- **Limitations** of local neighborhood features:
 - The metric is always zero if the two nodes do not have any neighbors in common.
 - However, the two nodes may still potentially be connected in the future.
- **Global neighborhood overlap metrics resolve the limitation by considering the entire graph.**



$$N_A \cap N_E = \emptyset$$

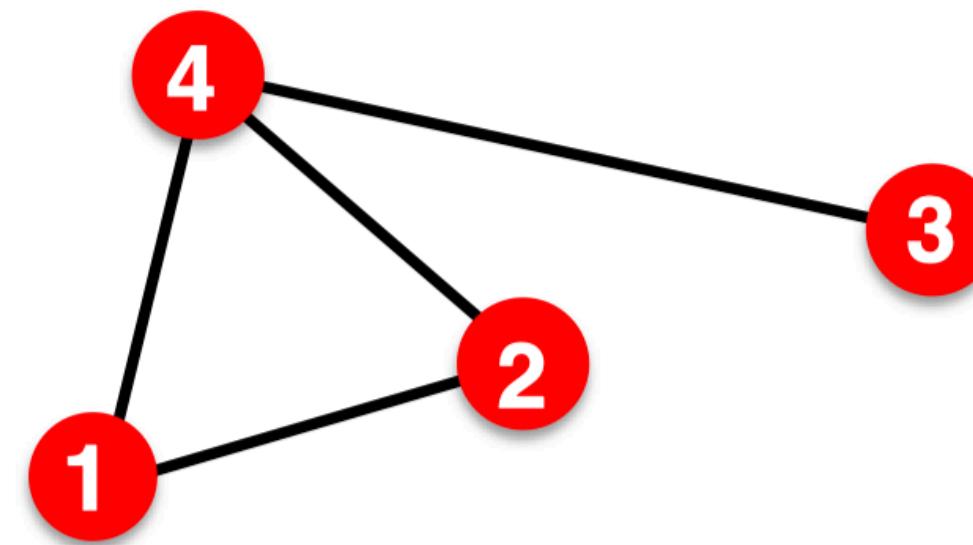
$$|N_A \cap N_E| = 0$$

Global Neighborhood Overlap

- Katz index: count the number of walks of all lengths between a given pair of nodes.
- Question: How to compute #walks between two nodes?
- Use powers of the graph adjacency matrix!

Intuition: Powers of Adj Matrices

- Computing #walks between two nodes
 - Recall: $A_{uv} = 1$ if $u \in N(v)$
 - Let $P_{uv}^{(K)} = \text{#walks of length } K \text{ between } u \text{ and } v$
 - We will show $P^{(K)} = A^k$
 - $P_{uv}^{(1)} = \text{#walks of length 1 (direct neighborhood) between } u \text{ and } v = A_{uv}$



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$P_{12}^{(1)} = A_{12}$

Intuition: Powers of Adj Matrices

- How to compute $P_{uv}^{(2)}$?
- Step 1: compute #walks of length 1 between each of u's neighbors and v
- Step 2: Sum up these #walks across u's neighbors

$$P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^1 = \sum_i A_{ui} * A_{iv} = A_{uv}^2$$

Node 1's neighbors #walks of length 1 between
Node 1's neighbors and Node 2

$$A^2 = \begin{matrix} \text{Power of} \\ \text{adjacency} \end{matrix} \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Global Neighborhood Overlap

- Katz index: count the number of walks of all lengths between a pair of nodes.
- How to compute #walks between two nodes?
- Use adjacency matrix powers!
 - A_{uv} specifies #walks of length 1 (direct neighborhood) between u and v
 - A_{uv}^2 specifies #walks of length 2 between u and v
 - ...
 - A_{uv}^l specifies #walks of length l between u and v

Global Neighborhood Overlap

- Katz index between v_1 and v_2 is calculated as the sum over all walks lengths:

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \beta^l A_{v_1 v_2}^l$$

$0 < \beta < 1$: discount factor (longer paths counts less)

- Katz index matrix is computed in closed form:

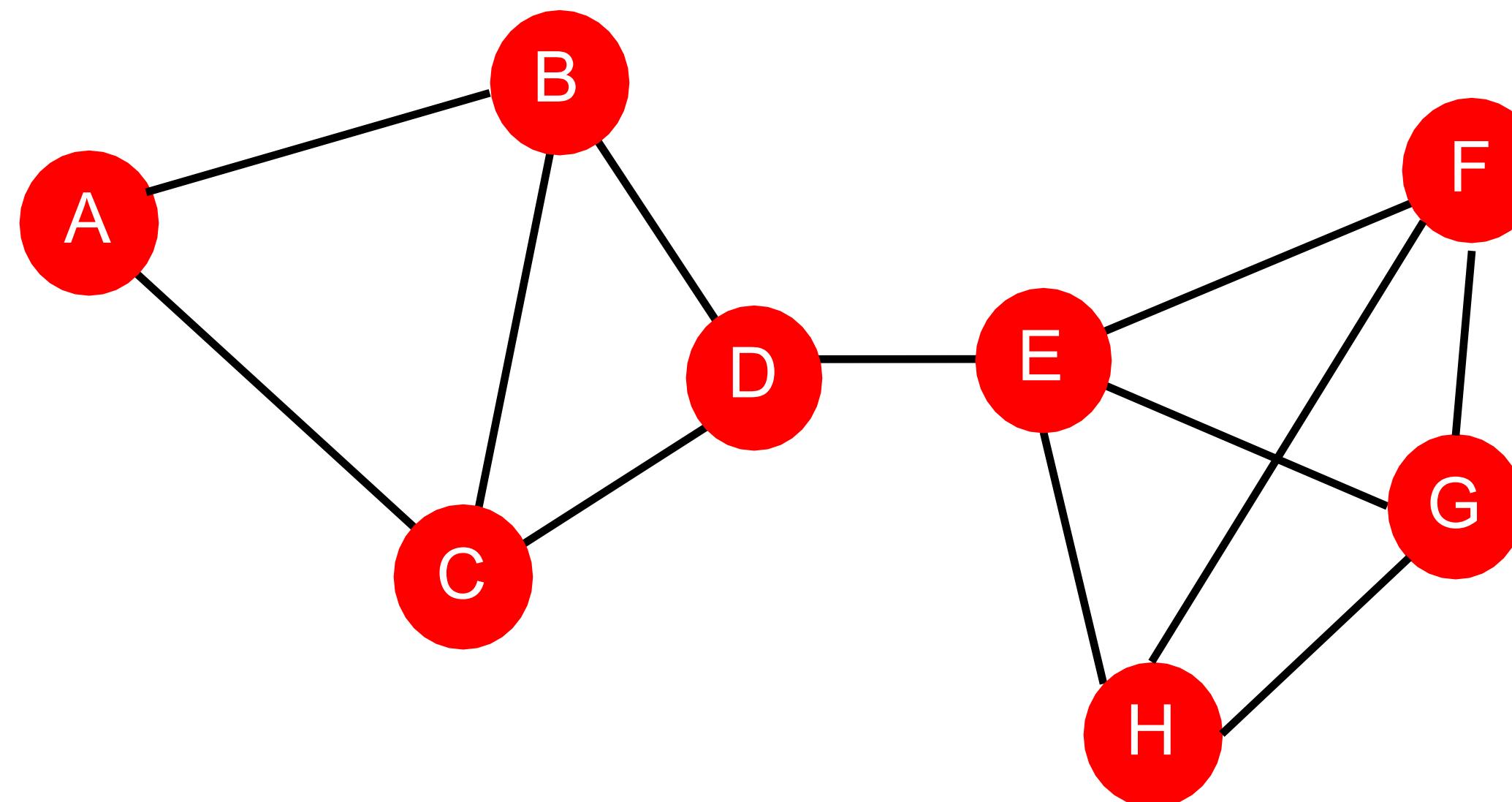
$$S = \sum_{i=1}^{\infty} \beta^i A^i = (I - \beta A)^{-1} - I \text{ by geometric series of matrices}$$

Link-Level Features: Summary

- **Distance-based features:**
 - It uses the shortest path length between two nodes but does not capture how the neighborhood overlaps.
- **Local neighborhood overlap:**
 - Captures how many neighboring nodes are shared by two nodes.
 - Becomes zero when no neighbor nodes are shared.
- **Global neighborhood overlap:**
 - It uses a global graph structure to score two nodes.
 - Katz index counts #walks of all lengths between two nodes.

Graph-Level Features

- **Goal:** We want features that characterize the structure of an entire graph.
- For example:



Background: Kernel Methods

- Kernel methods are widely used for traditional ML for graph-level prediction.
- **Idea:** Design kernels instead of feature vectors.
- A quick introduction to Kernels:
 - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
 - Kernel matrix $\kappa = (K(G, G'))_{G,G'}$ must always be positive semidefinite (i.e., has positive eigenvalues)
 - There exists a feature representation $\Phi(\cdot)$ such that $K(G, G') = \Phi(G)^T \Phi(G')$
 - Once the kernel is defined, an off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

Graph-Level Features: Overview

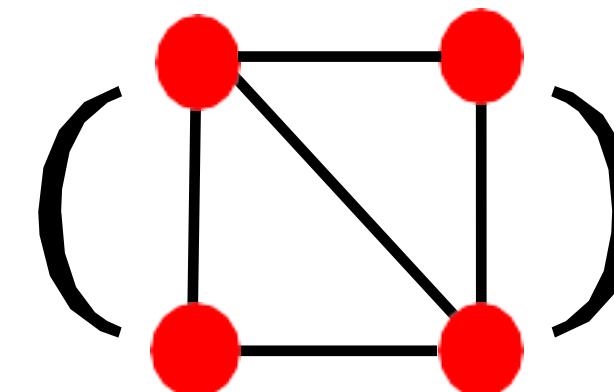
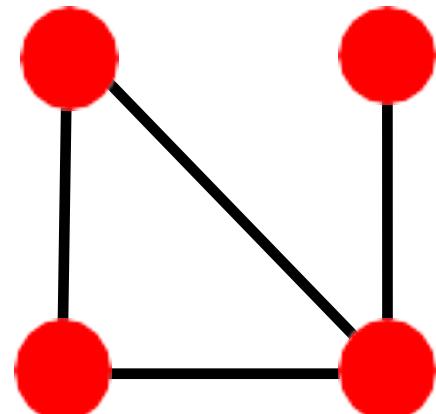
Graph Kernels:

- Measure the similarity between two graphs:
 - Graphlet Kernel [1]
 - Weisfeiler-Lehman Kernel [2]
 - Other kernels are also proposed in the literature
 - (beyond the scope of this lecture)
 - Random-walk kernel
 - Shortest-path graph kernel
 - And many more...

1. Shervashidze, Nino, et al. "Efficient graphlet kernels for large graph comparison." Artificial Intelligence and Statistics. 2009.
2. Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." Journal of Machine Learning Research 12.9 (2011).

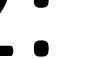
Graph Kernel: Key Idea

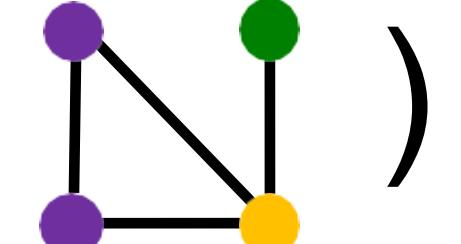
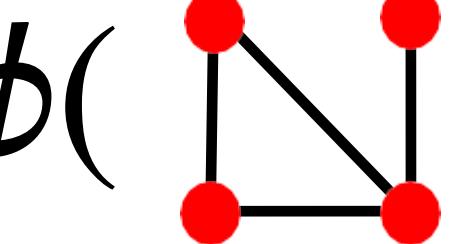
- Goal: Design graph feature vector $\Phi(G)$
- Key idea: Bag-of-Words (BoW) for a graph
 - Recall: BoW uses the word counts as document features (no ordering considered).
 - Naïve extension to a graph: regard nodes as words.
 - Since both graphs have 4 red nodes, we get the same feature vector for two different graphs...



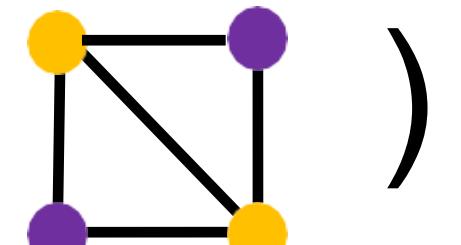
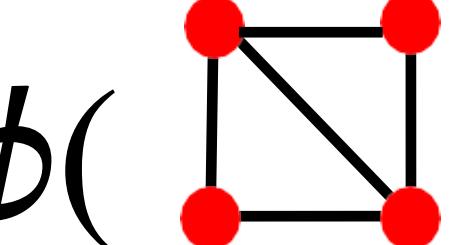
Graph Kernel: Key Idea

What if we use **Bag of node degrees?**

Deg1:  Deg2:  Deg3: 

$$\phi(\text{graph}) = \text{count}(\text{graph}) = [1, 2, 1]$$


Obtains different features for
different graphs!

$$\phi(\text{graph}) = \text{count}(\text{graph}) = [0, 2, 2]$$


Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use Bag-of-* representation of graph, where * is more sophisticated than node degrees!

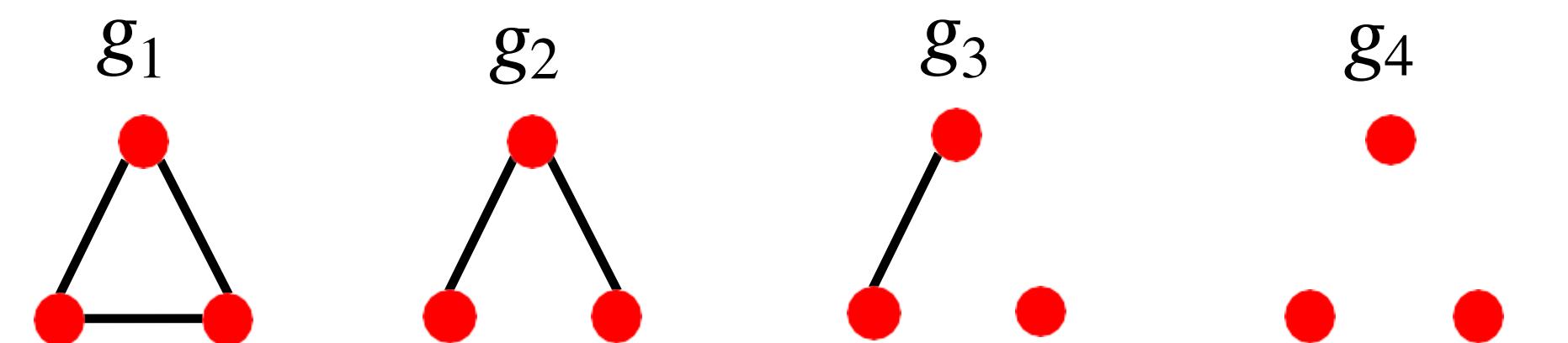
Graph-Level Graphlet Features

- **Key idea:** Count the number of different graphlets in a graph.
 - Note: The definition of graphlets here differs slightly from the node-level features.
 - The two differences are:
 - Nodes in graphlets here do not need to be connected (allows for isolated nodes)
 - The graphlets here are not rooted.
 - Examples in the next slide illustrate this.

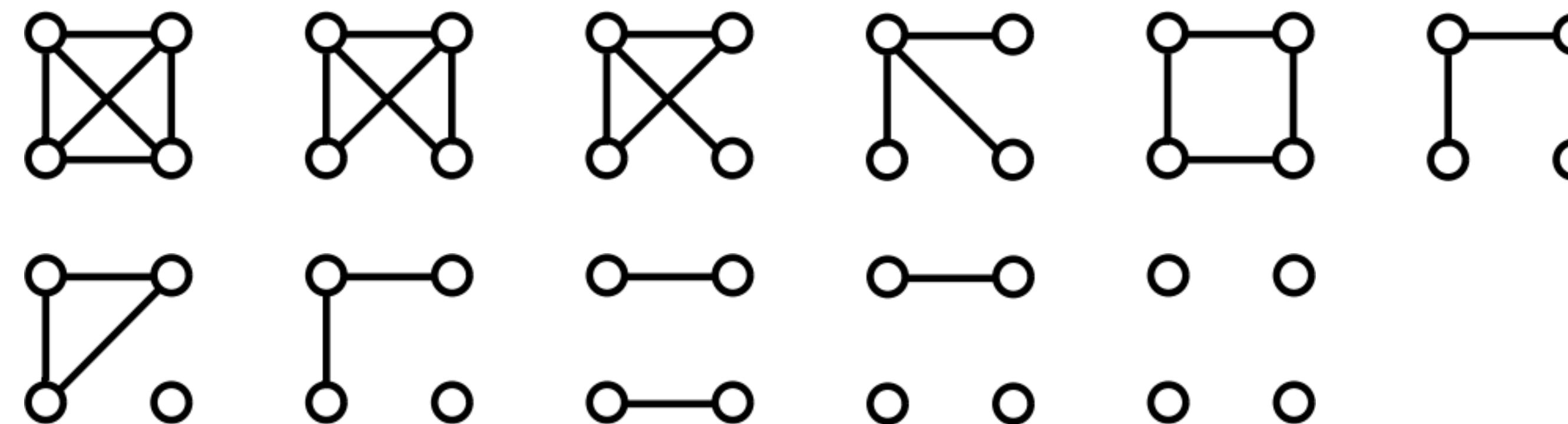
Graph-Level Graphlet Features

Let $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$ be a list of graphlets of size k .

- For $k = 3$, there are 4 graphlets.



- For $k = 4$, there are 11 graphlets.



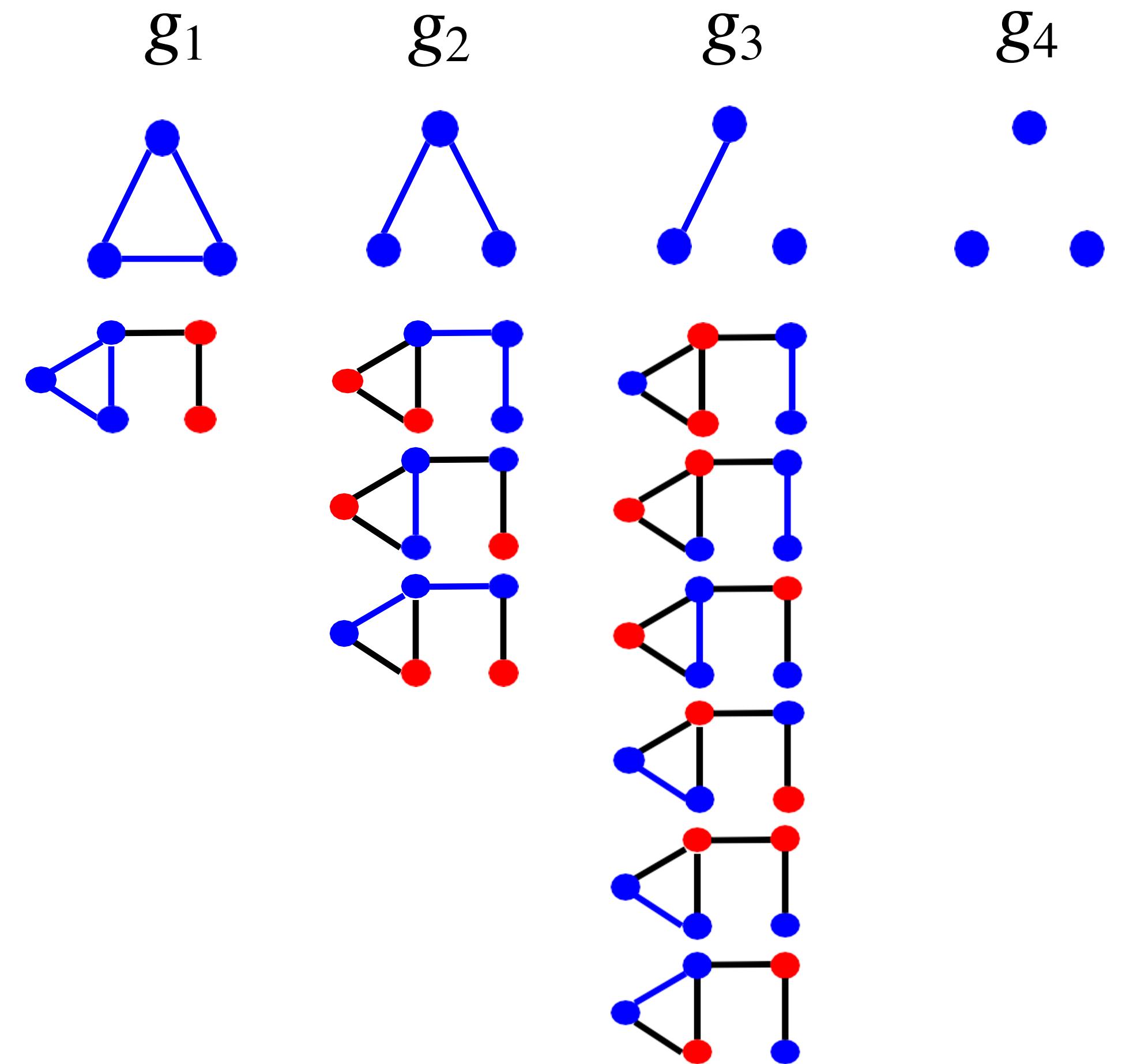
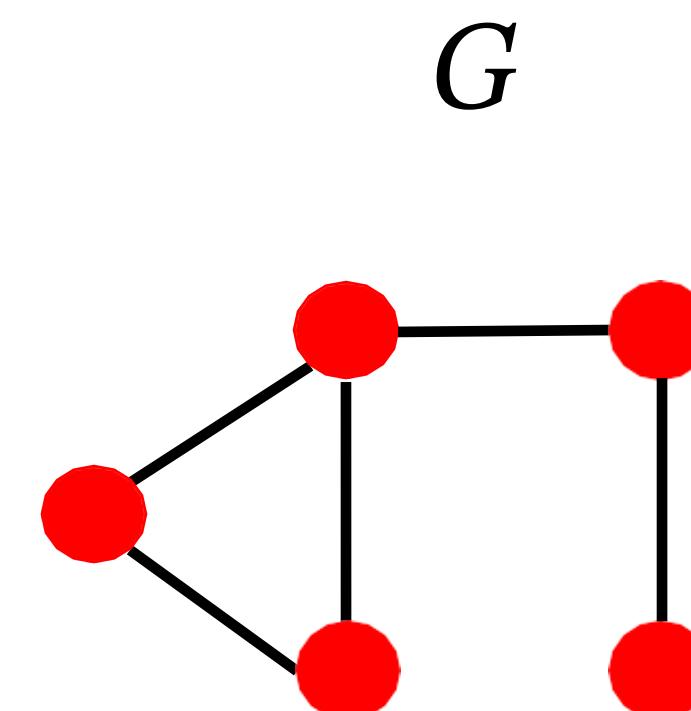
Graph-Level Graphlet Features

Given graph G and a graphlet list $G_k = (g_1, g_2, \dots, g_{n_k})$
define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

$$f(G)_i = \#(g_i \subseteq G) \text{ for } i = 1, 2, \dots, n_k$$

Graph-Level Graphlet Features

Example for $k = 3$.



$$f_G = (1, \quad 3, \quad 6, \quad 0)^T$$

Graph-Level Graphlet Kernel

Given two graphs, G and G' , graphlet kernel is computed as

$$K(G, G') = f(G)^T f(G')$$

Problem: if G and G' have different sizes, that will greatly skew the value.

Solution: normalize each feature vector

$$h(G) = \frac{f_G}{\text{sum}(f_G)}$$

The Graphlet Kernel

- Limitations: **Counting graphlets is expensive!**
- Counting size- k graphlets for a graph with size n by enumeration takes n^k .
- This is unavoidable in the worst case since the subgraph isomorphism test (judging whether a graph is a subgraph of another graph) is **NP-hard**.
- If a graph's node degree is bounded by d , an $O(nd^{k-1})$ algorithm exists to count all the graphlets of size k .
- **Can we design a more efficient graph kernel?**

Weisfeiler-Lehman Kernel

- **Goal**: Design an efficient graph feature descriptor $\Phi(G)$
- **Idea**: Use neighborhood structure to enrich node vocabulary iteratively.
 - Generalized version of Bag of node degrees since node degrees are one-hop neighborhood information.
- Algorithm to achieve this:
 - **Color refinement**

Color Refinement

Given: A graph G with a set of nodes V .

- Assign an initial color $c^{(0)}(v)$ to each node v .
- Iteratively refine node colors by

$$c^{k+1}(v) = \text{HASH}(c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)})$$

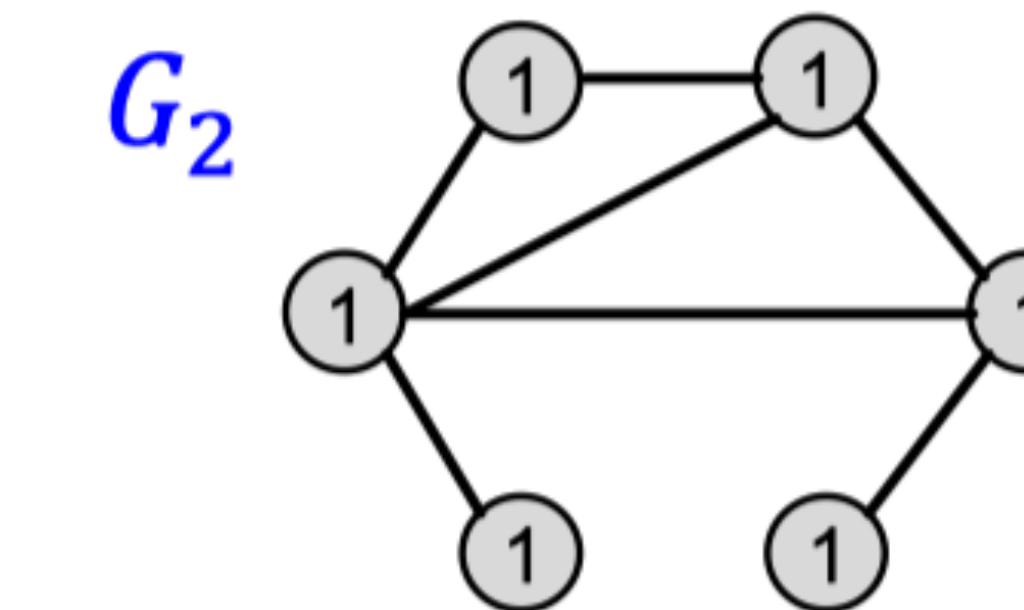
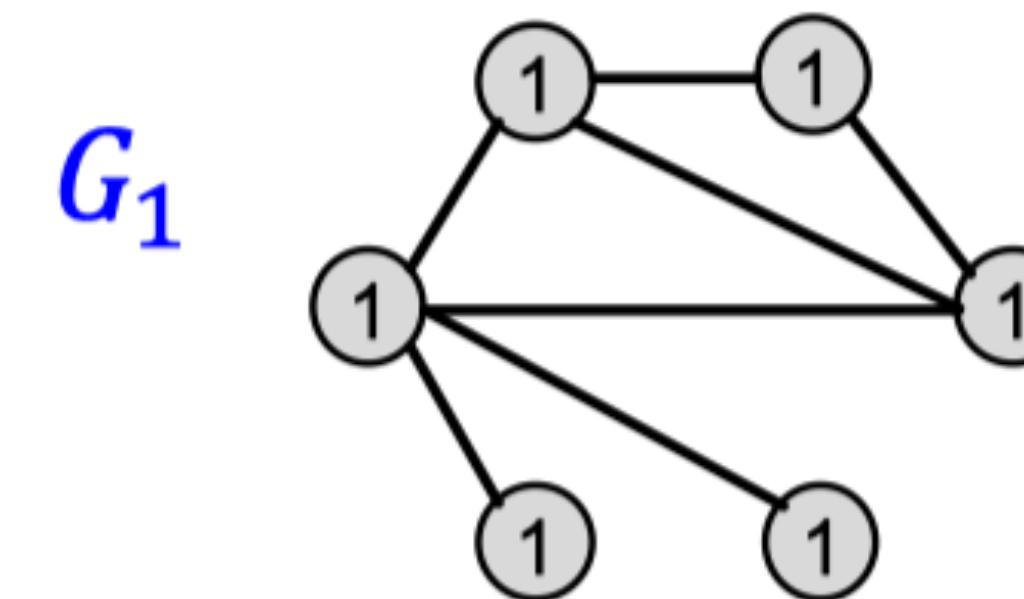
where HASH maps different inputs to different colors

- After k steps of color refinement, $c^{(K)}(v)$ summarizes the structure of the K-hop neighborhood

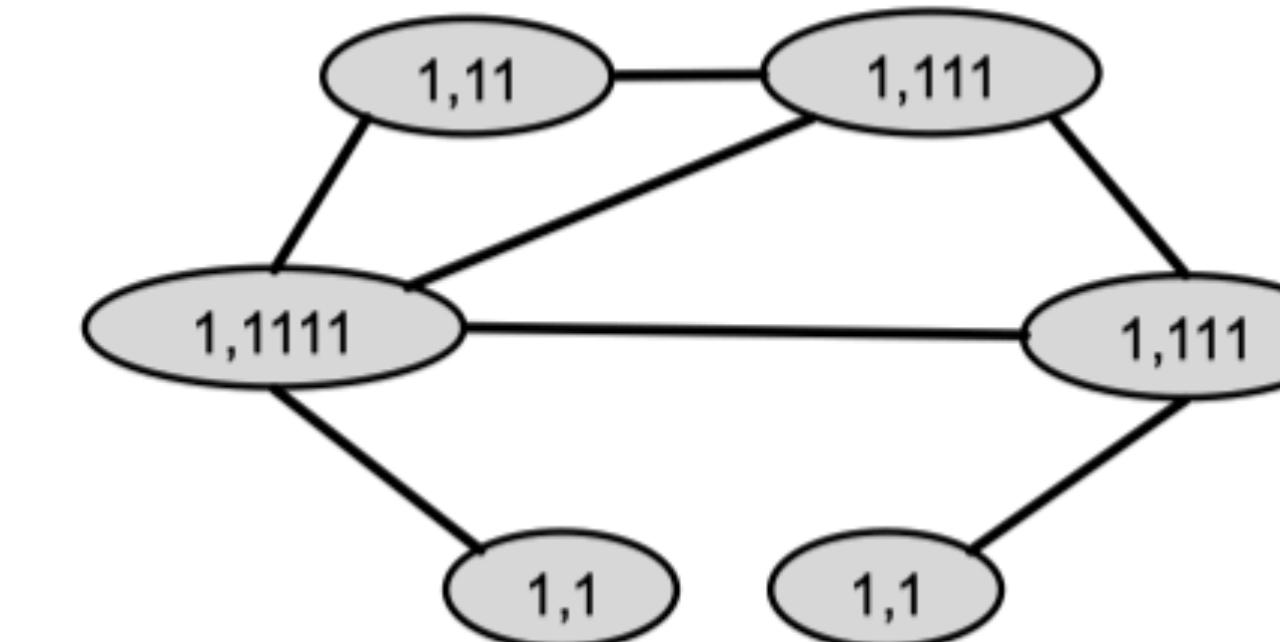
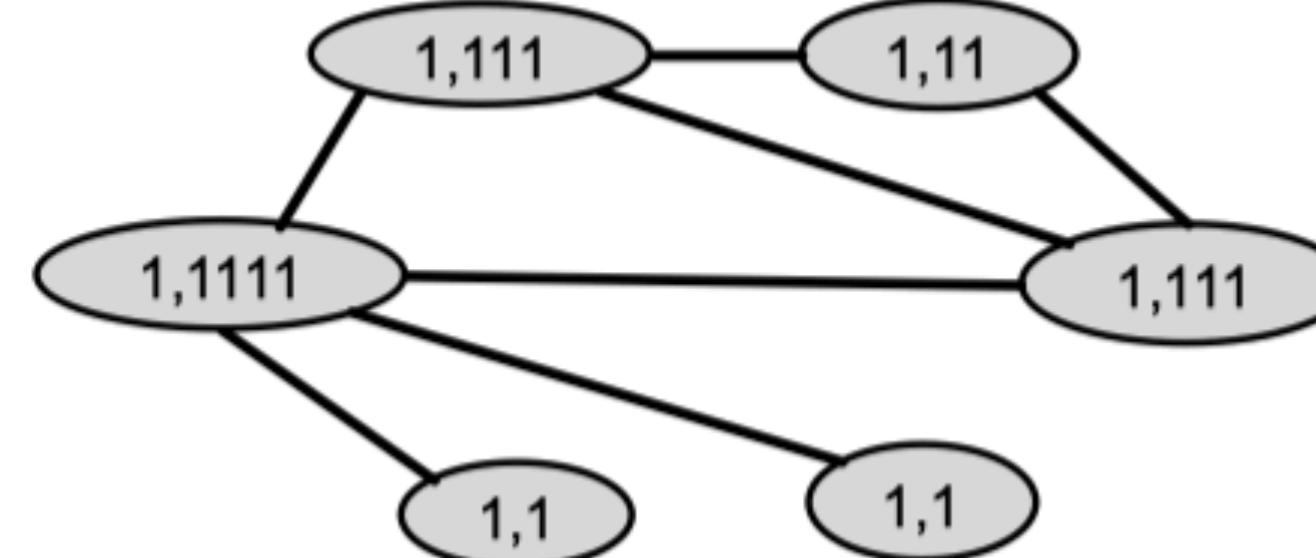
Color Refinement (1)

Example of color refinement given two graphs

- Assign initial colors



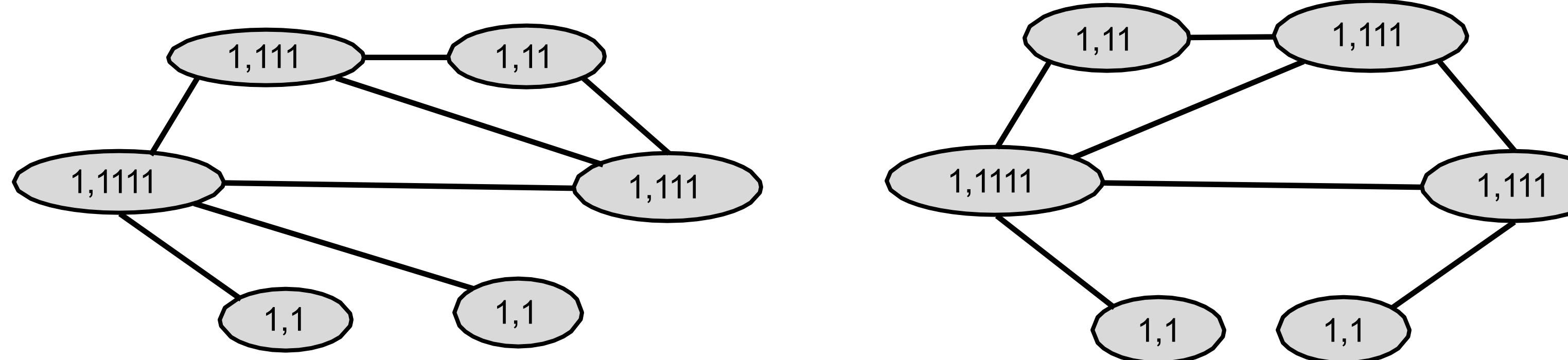
- Aggregate neighboring colors



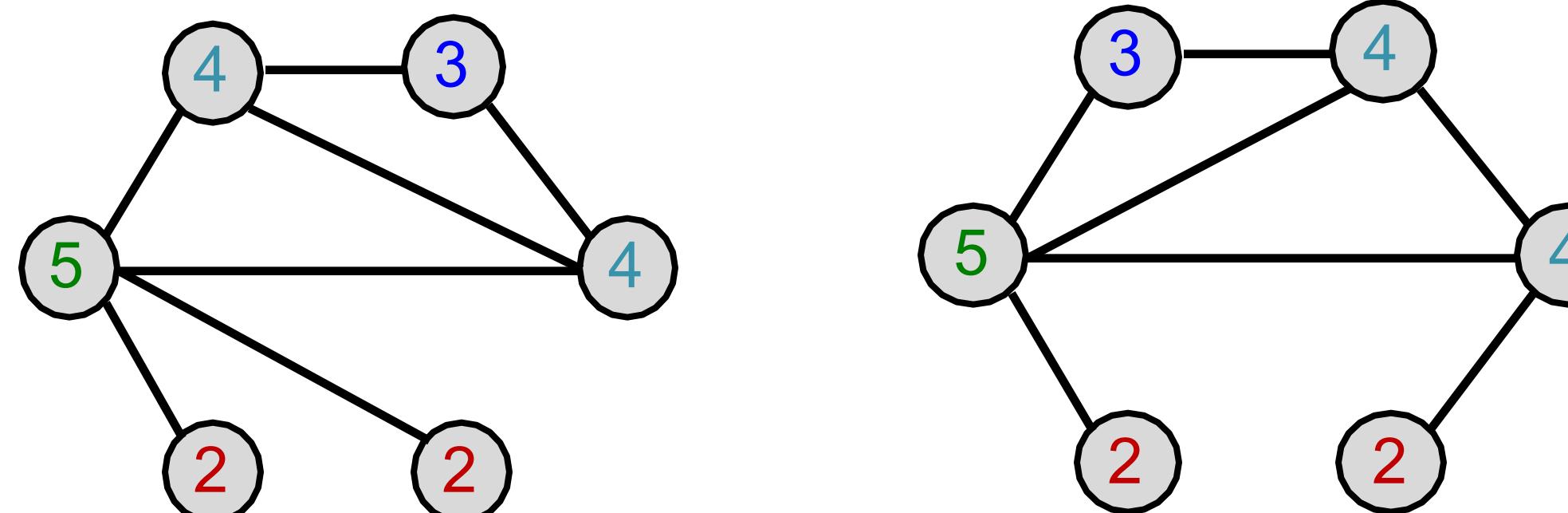
Color Refinement (2)

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors



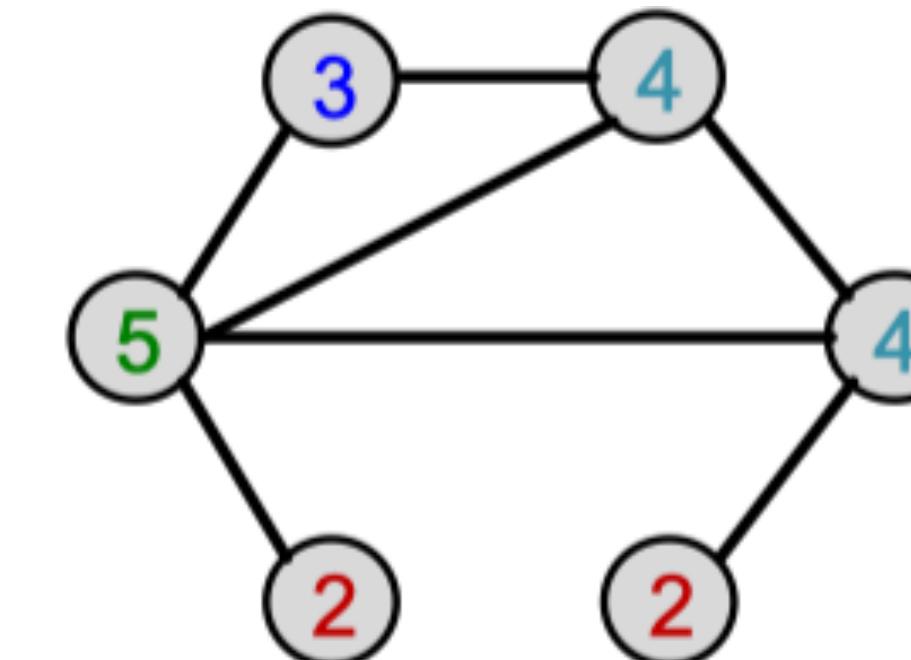
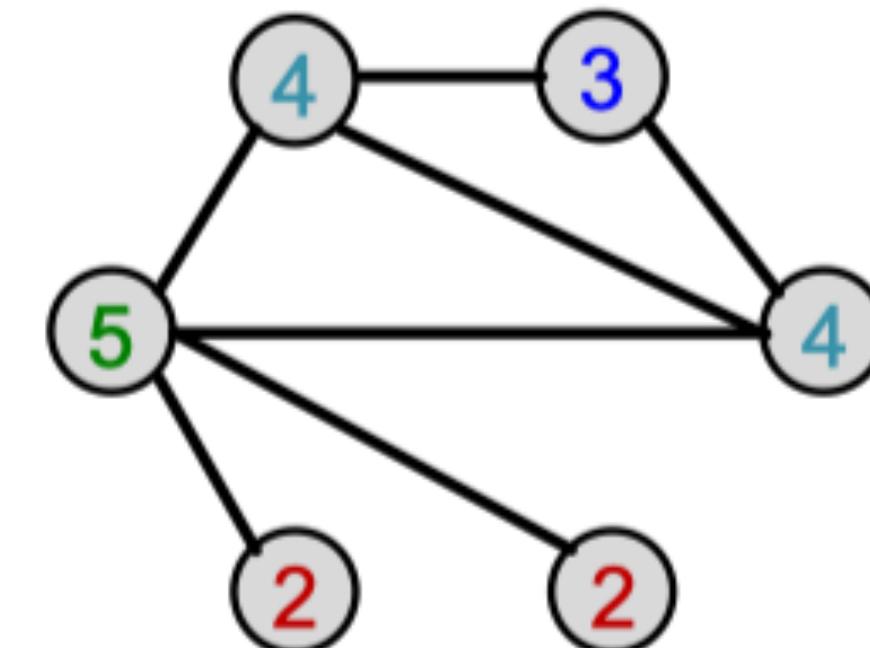
Hash table

1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

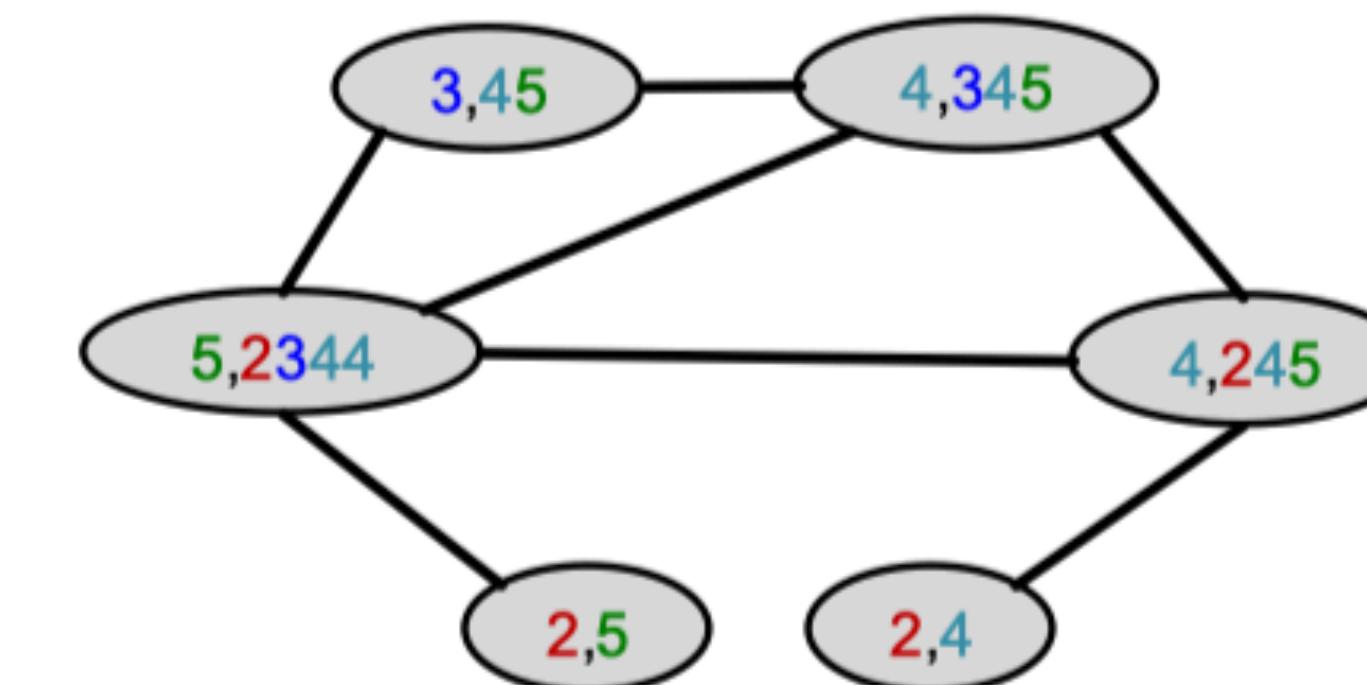
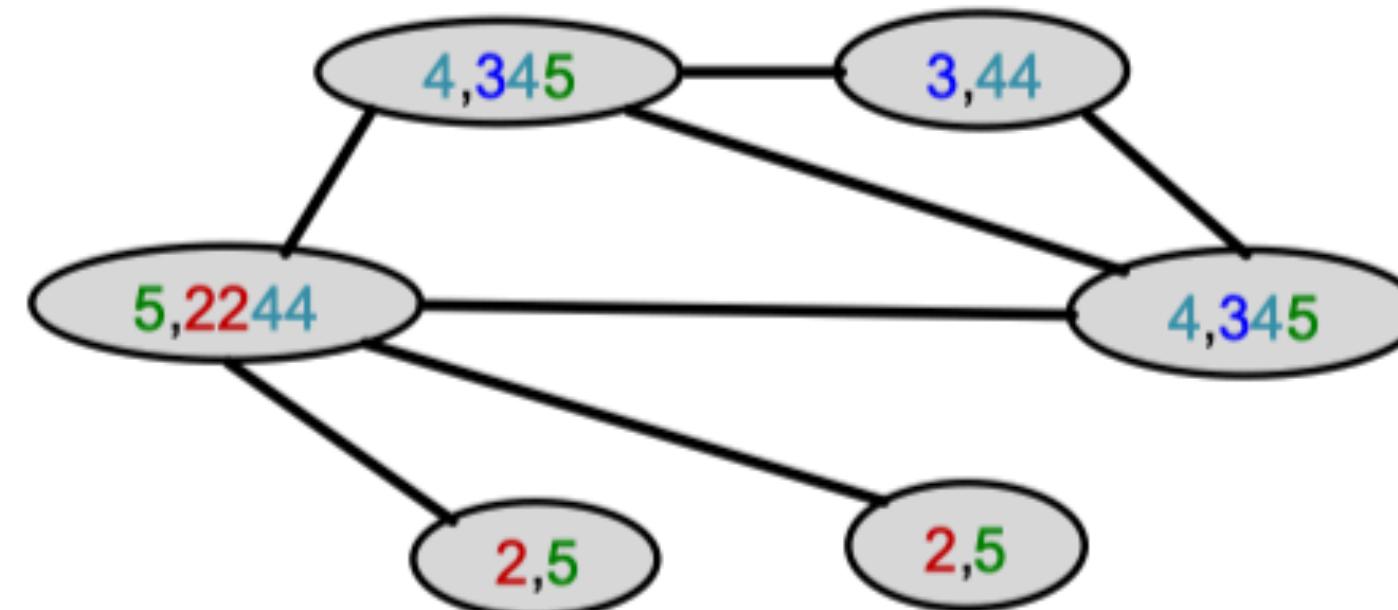
Color Refinement (3)

Example of color refinement given two graphs

- Aggregated colors



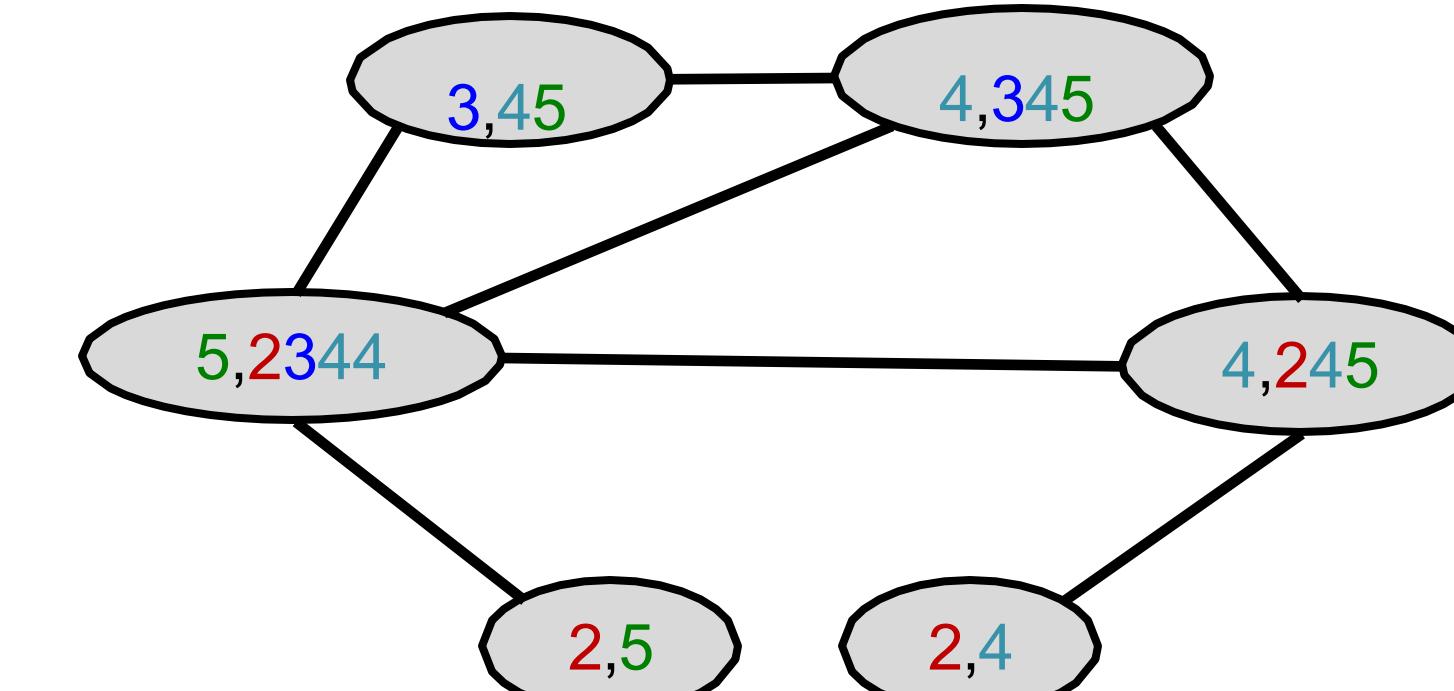
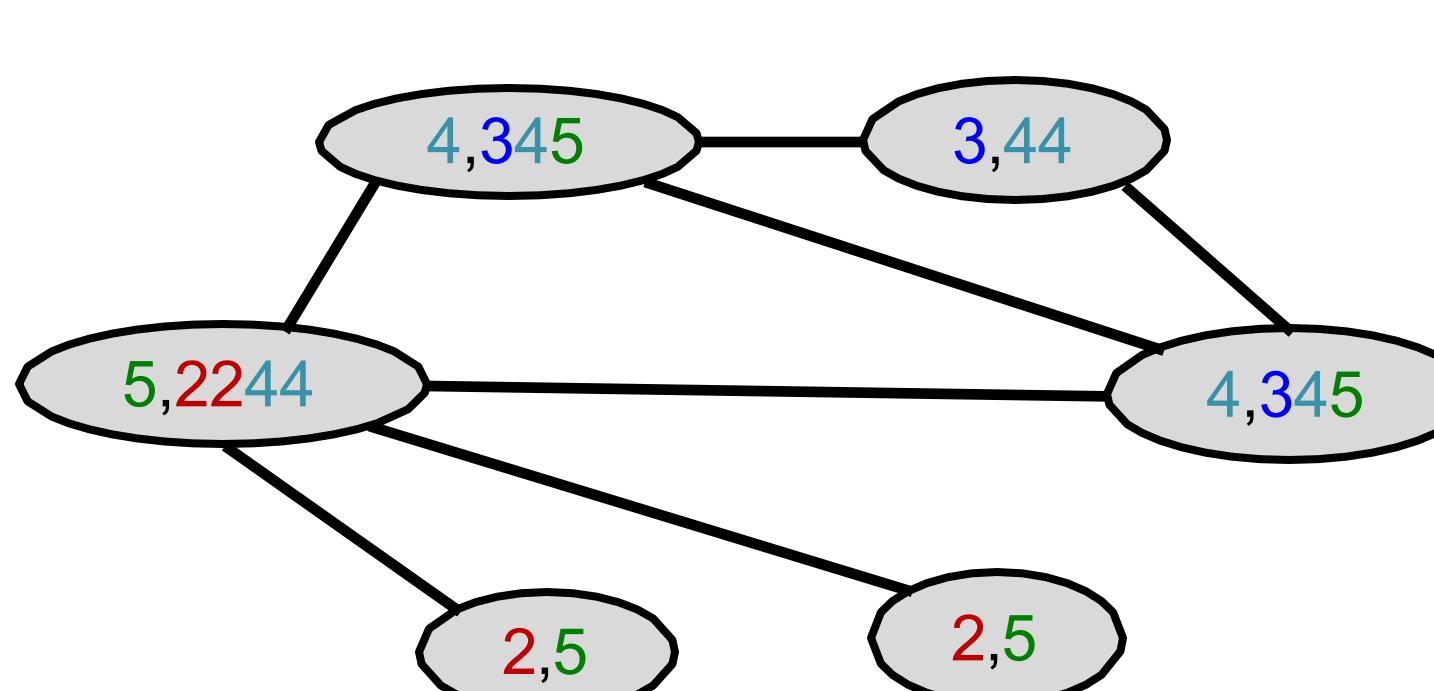
- Hash aggregated colors



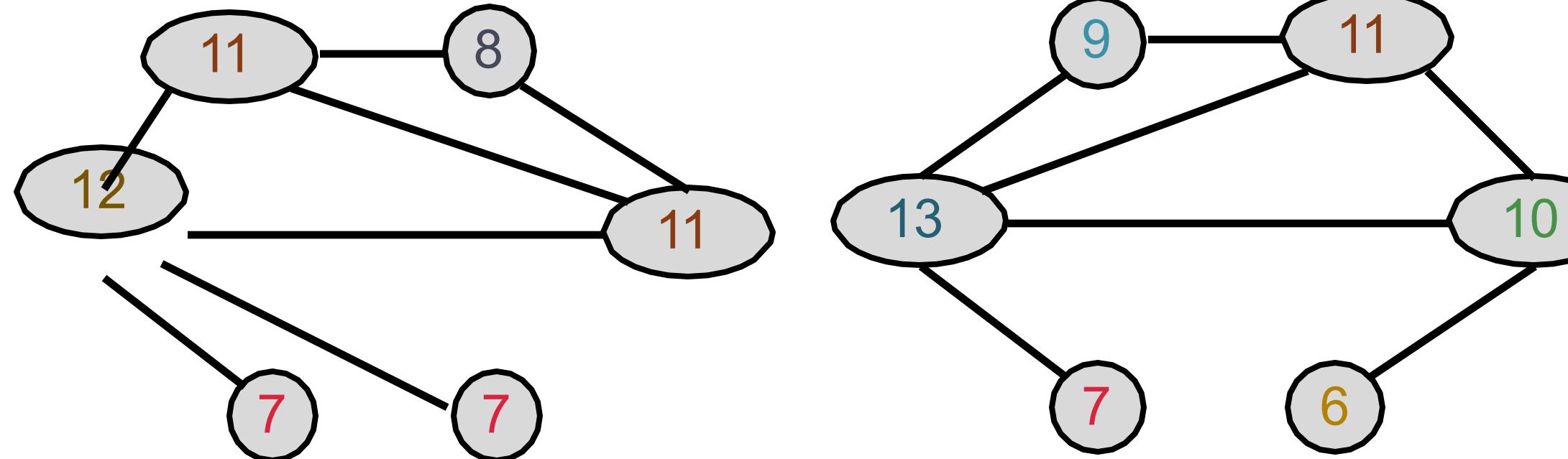
Color Refinement (4)

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors

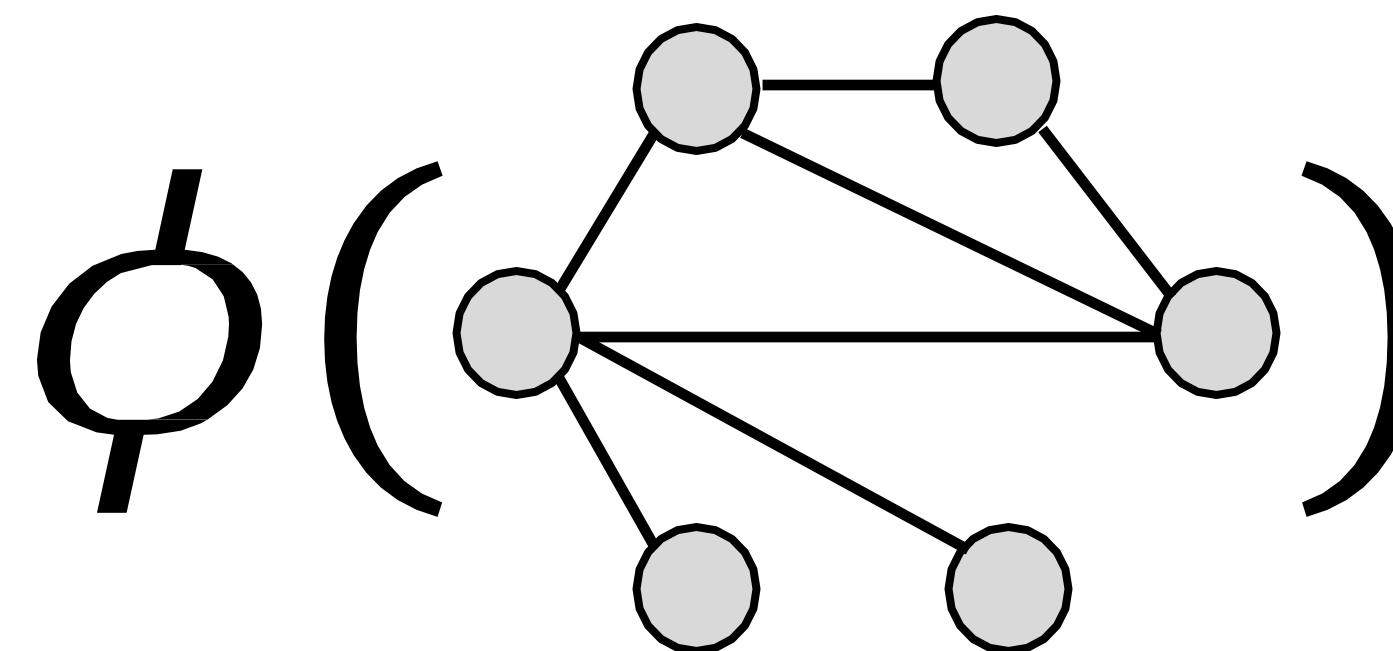


Hash table

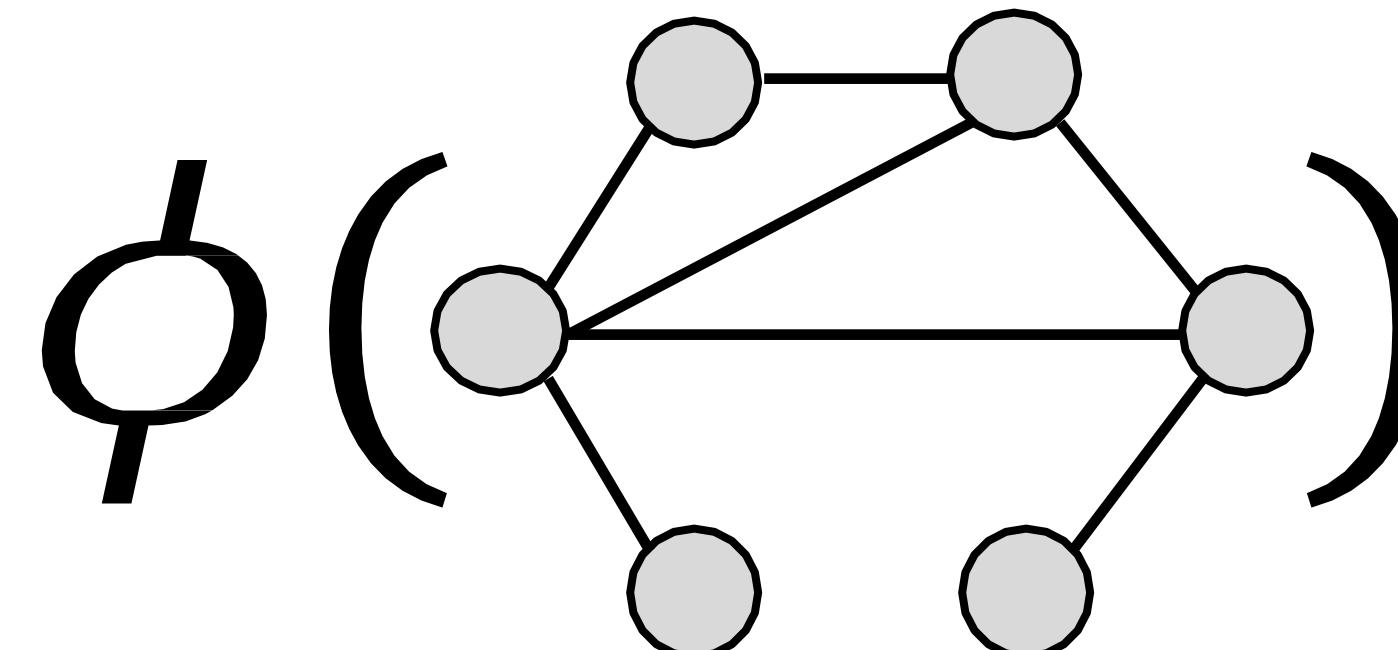
2,4	-->	6
2,5	-->	7
3,44	-->	8
3,45	-->	9
4,245	-->	10
4,345	-->	11
5,2244	-->	12
5,2344	-->	13

Weisfeiler-Lehman Graph Features

After color refinement, WL kernel counts number of nodes with a given color.



Colors
1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,0,2,1,0,0,2,1,0]
Counts



$$1, \textcolor{red}{2}, \textcolor{blue}{3}, \textcolor{teal}{4}, \textcolor{green}{5}, \textcolor{brown}{6}, \textcolor{red}{7}, \textcolor{darkgray}{8}, \textcolor{teal}{9}, \textcolor{green}{10}, \textcolor{brown}{11}, \textcolor{brown}{12}, \textcolor{teal}{13} \\ = [6, 2, 1, 2, 1, 1, 1, 0, 1, 1, 1, 0, 1]$$

Weisfeiler-Lehman Kernel

The WL kernel value is computed by the inner product of the color count vectors:

$$\begin{aligned} K(& \text{graph}_1, \text{graph}_2) \\ &= \phi(\text{graph}_1)^T \phi(\text{graph}_2) \\ &= 49 \end{aligned}$$

Weisfeiler-Lehman Kernel

- **WL kernel is computationally efficient.**
 - The time complexity for color refinement at each step is linear in #(edges) since it aggregates neighboring colors.
 - When computing a kernel value, only colors appearing in the two graphs must be tracked.
 - Thus, at most, #(colors) is the total number of nodes.
 - Counting colors takes linear time w.r.t. #(nodes).
 - In total, **time complexity is linear in #(edges)**.

Graph-level Features: Summary

- **Graphlet Kernel**
 - Graph is represented as Bag-of-graphlets
 - Computationally expensive
- **Weisfeiler-Lehman Kernel**
 - Apply K -step color refinement algorithm to enrich node colors
 - Different colors capture different K -hop neighborhood structures
 - Graph is represented as Bag-of-colors
 - Computationally efficient
 - Closely related to Graph Neural Networks (for the next course!)

Summary

- **Traditional ML Pipeline**
 - Hand-crafted (structural) features + ML model
- **Hand-crafted features for graph data**
 - **Node-level:**
 - Node degree, centrality, clustering coefficient, graphlets
 - **Link-level:**
 - Distance-based feature
 - local/global neighborhood overlap
 - **Graph-level:**
 - Graphlet kernel, WL kernel
- However, we only considered featurizing the graph structure (but not the attribute of nodes and their neighbors)



Q&A

