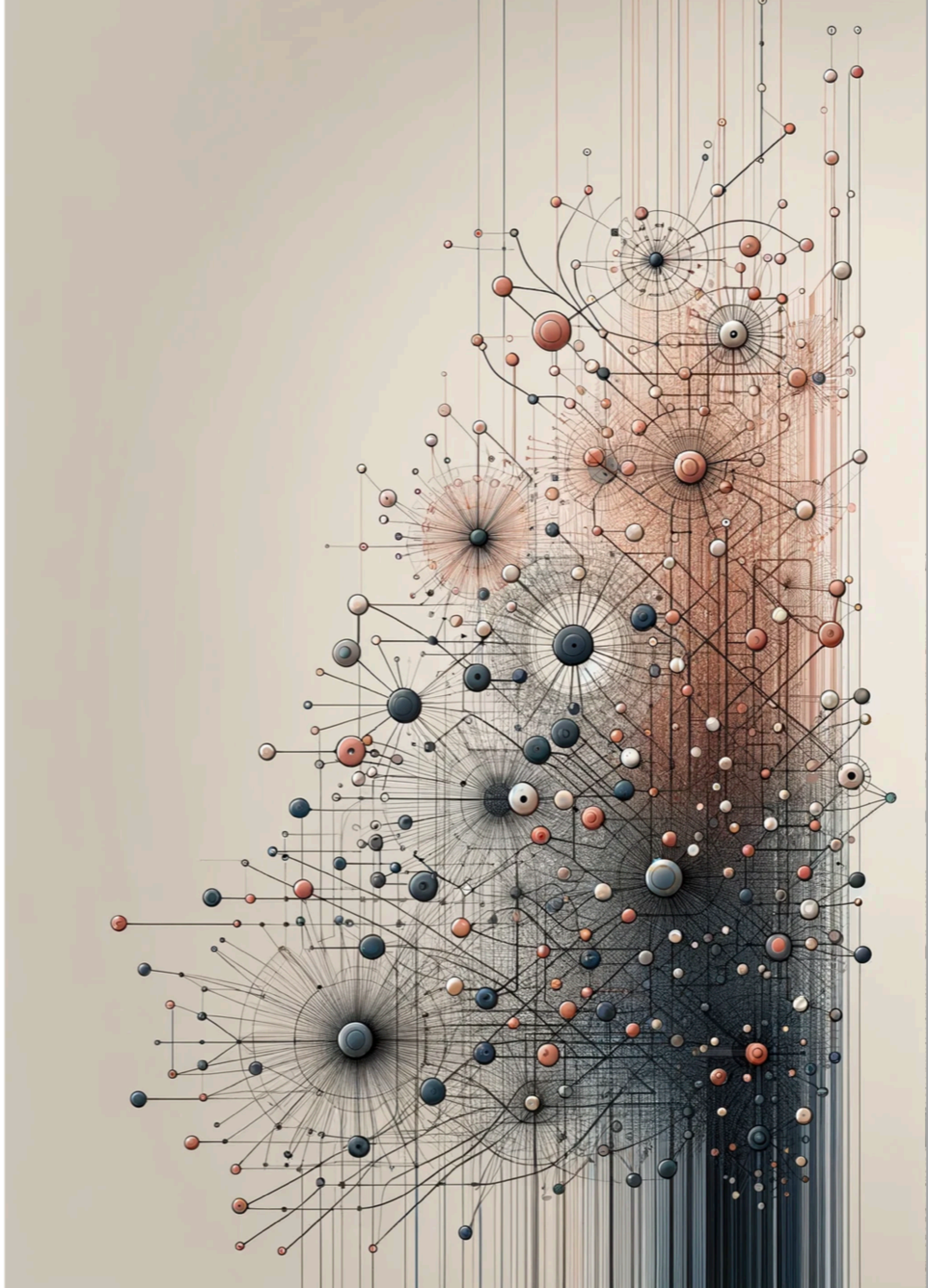
UNIVERSITÀ DI TORINO

Analisi e Visualizzazione delle Reti Complesse

NS22 - Traditional Machine Learning on Graphs

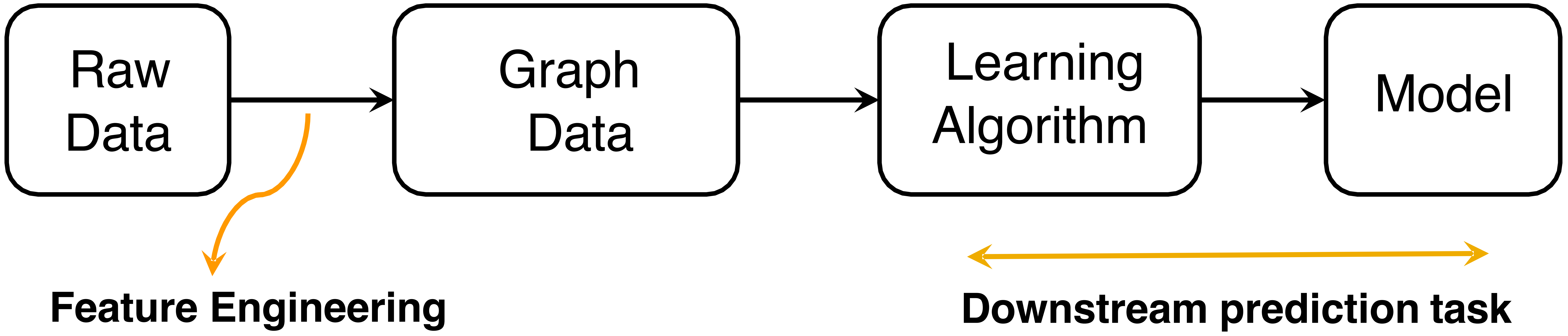
Prof. Rossano Schifanella



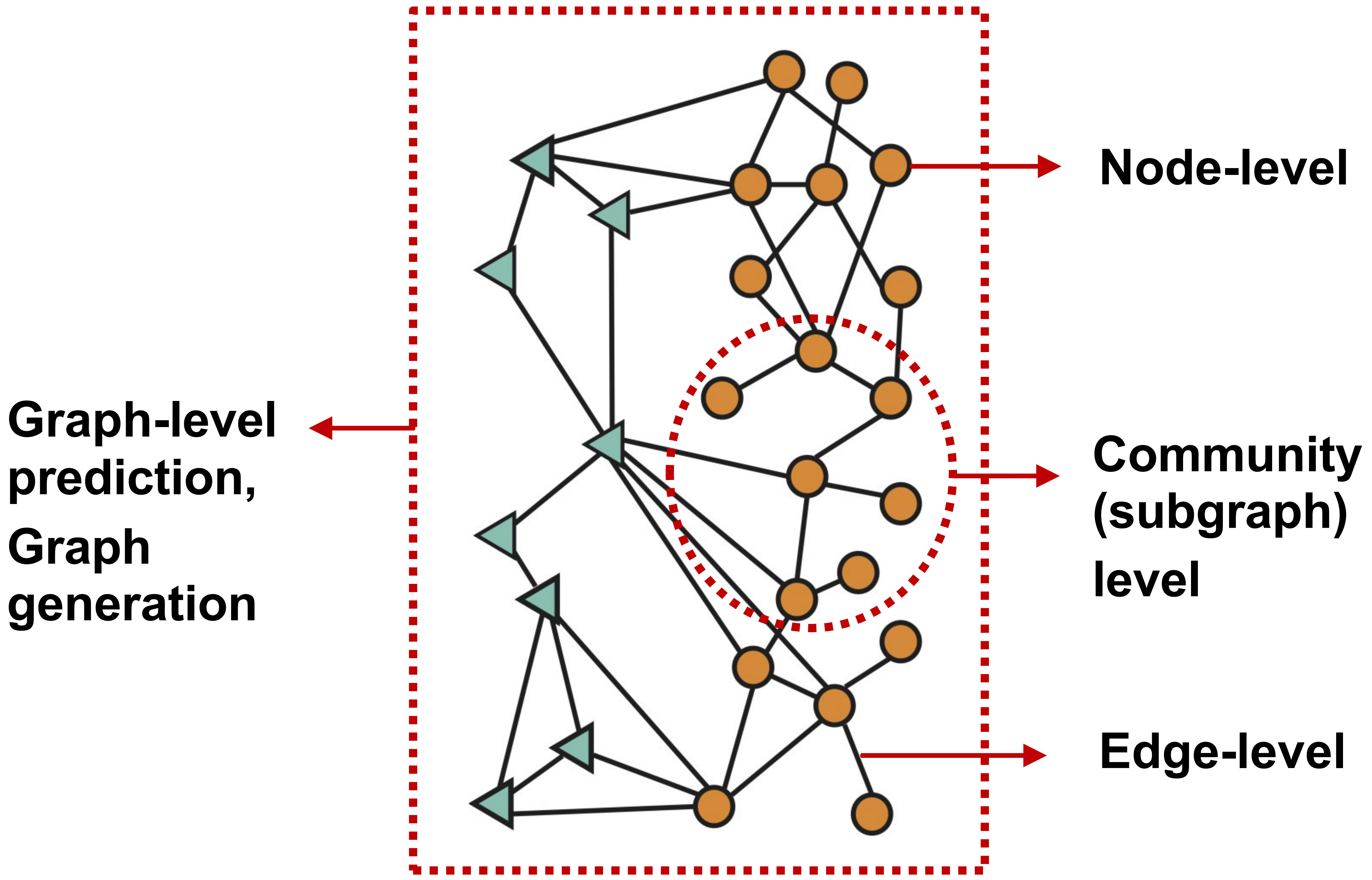
General Pipeline

(Supervised) Machine Learning Lifecycle

This feature, that feature. Every single time!



Different types of tasks

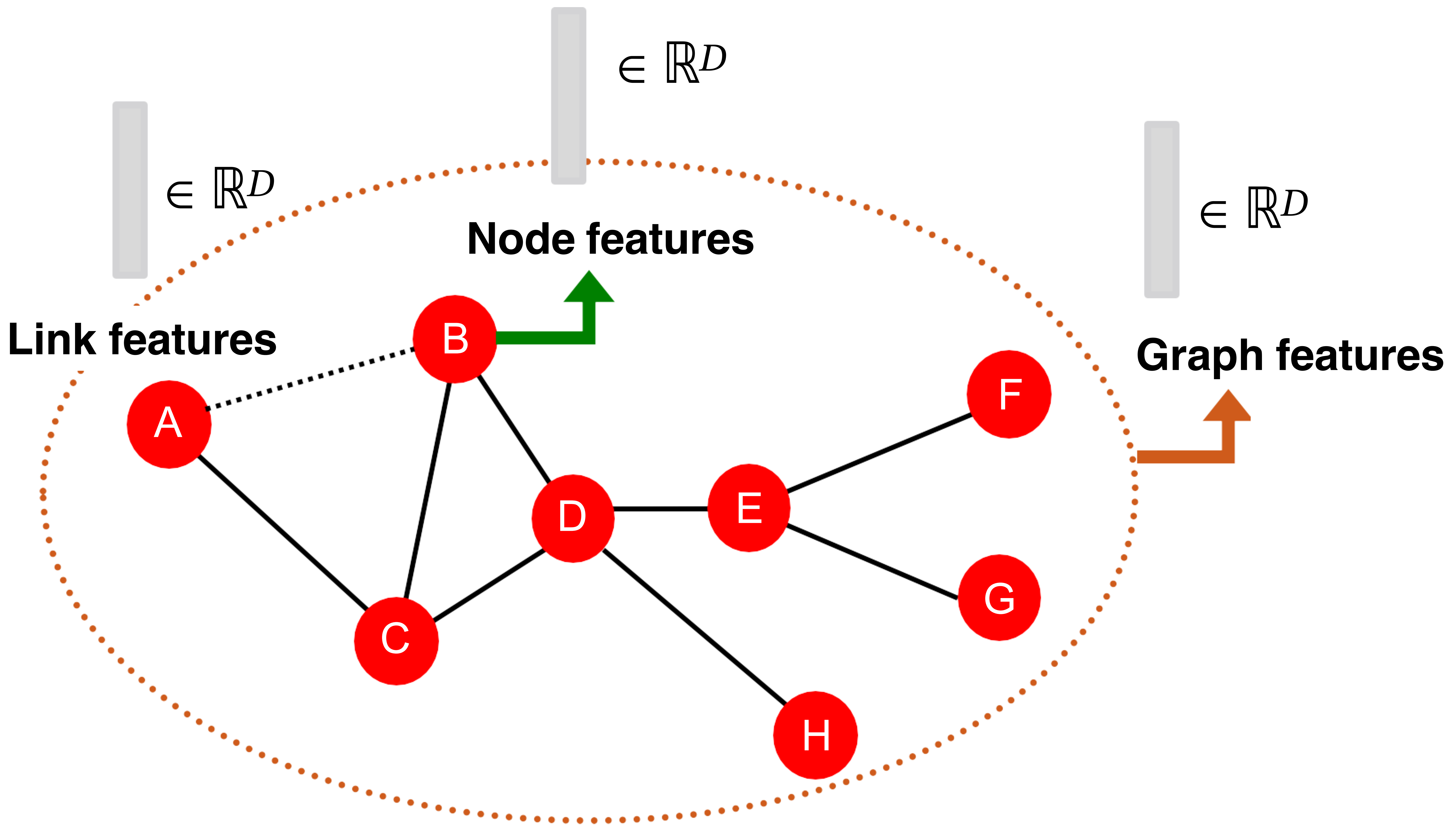

Node-level
Graph-level prediction, Graph generation
Community (subgraph) level
Edge-level

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

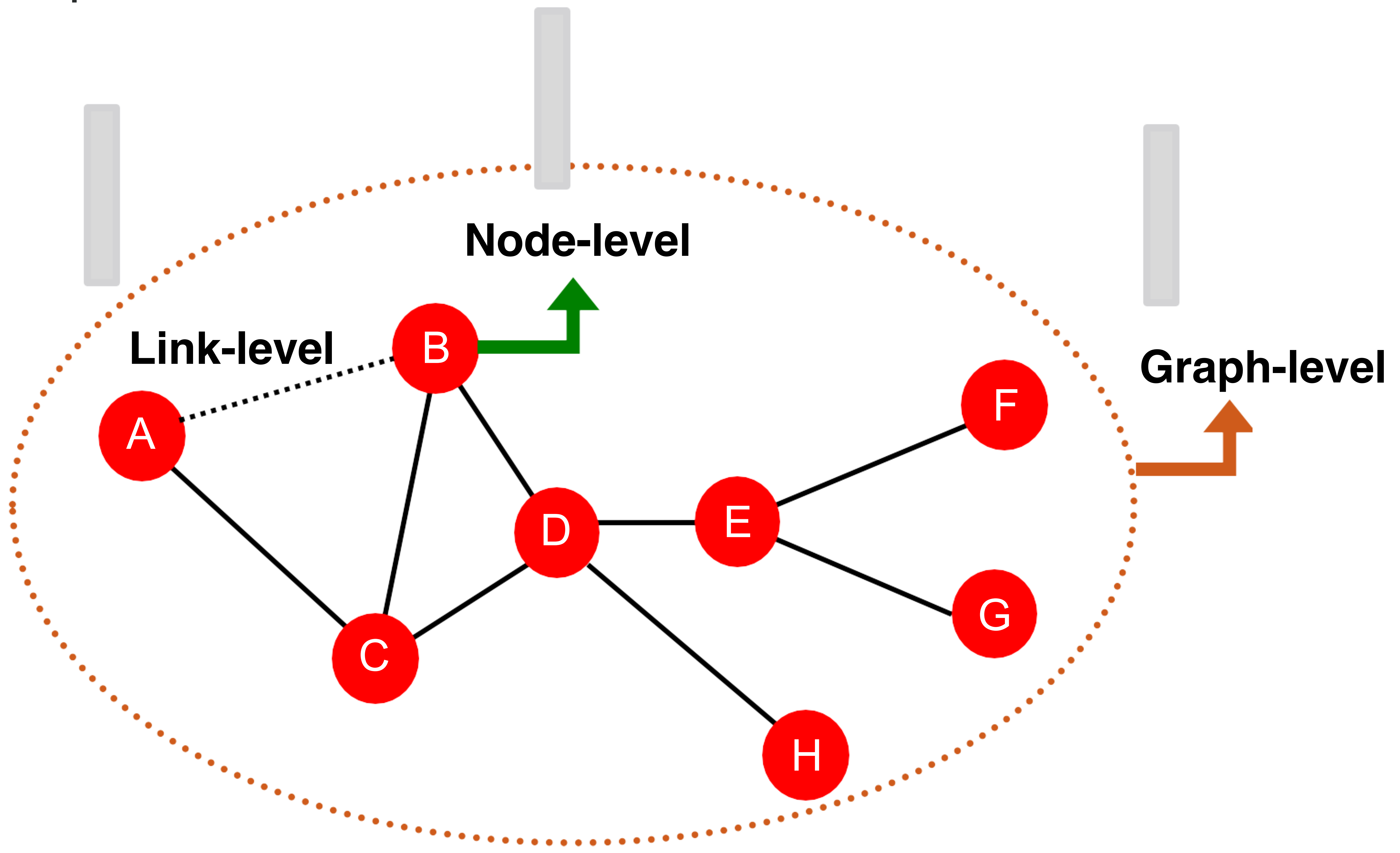
Traditional LM Pipeline

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



Machine Learning Tasks: Review

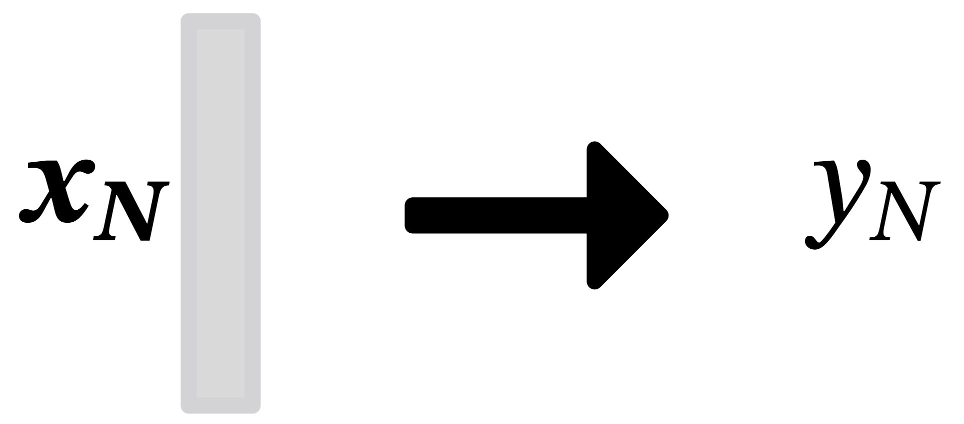
- **Node-level** prediction
- **Link-level** prediction
- **Graph-level** prediction

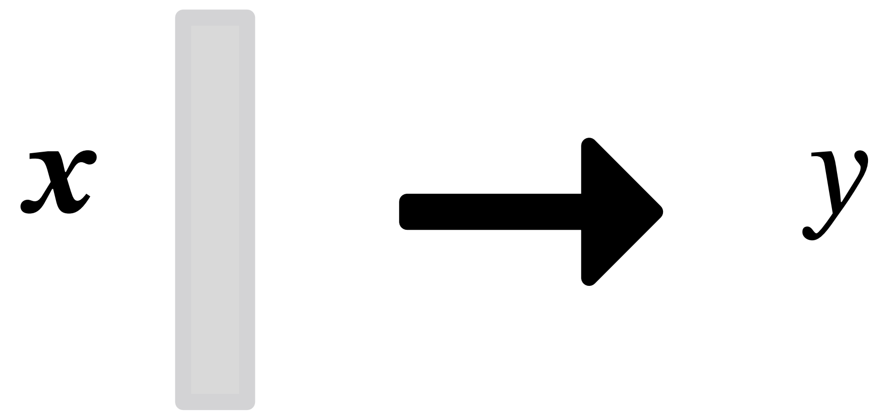


Traditional LM Pipeline

- **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





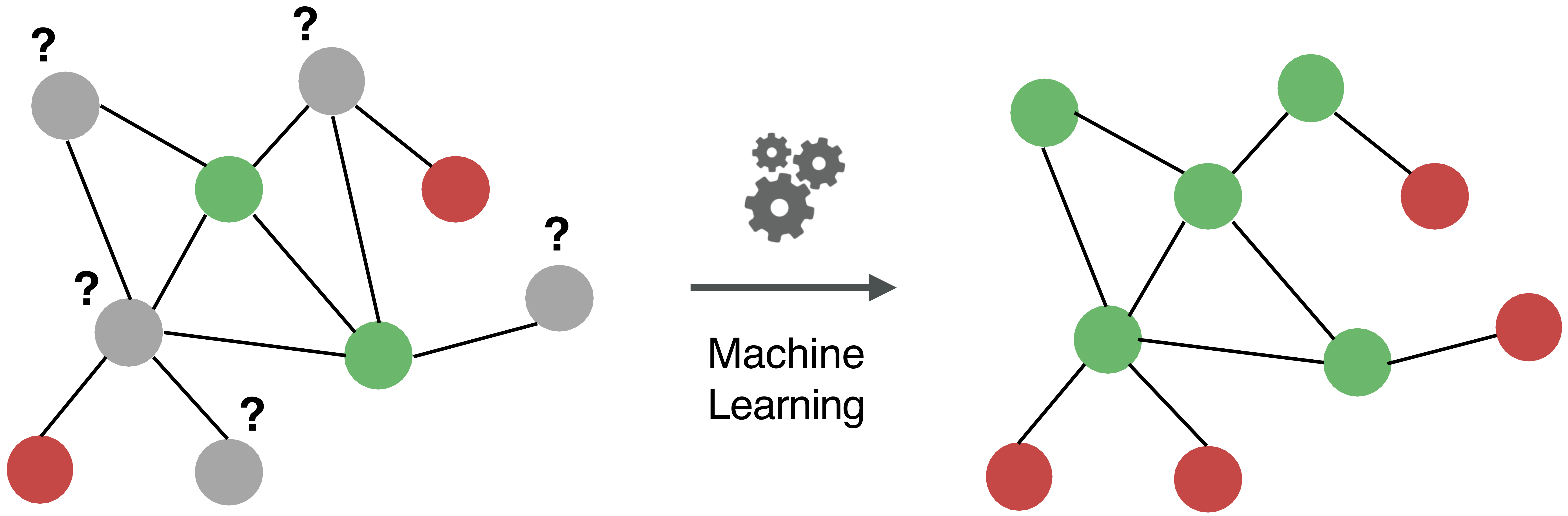
Features Design

- 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 $\boldsymbol{x}$
  - **Objects**: Nodes, edges, sets of nodes, entire graphs
  - **Objective function**:
    - What task are we aiming to solve?

Node-Level Tasks



Node classification

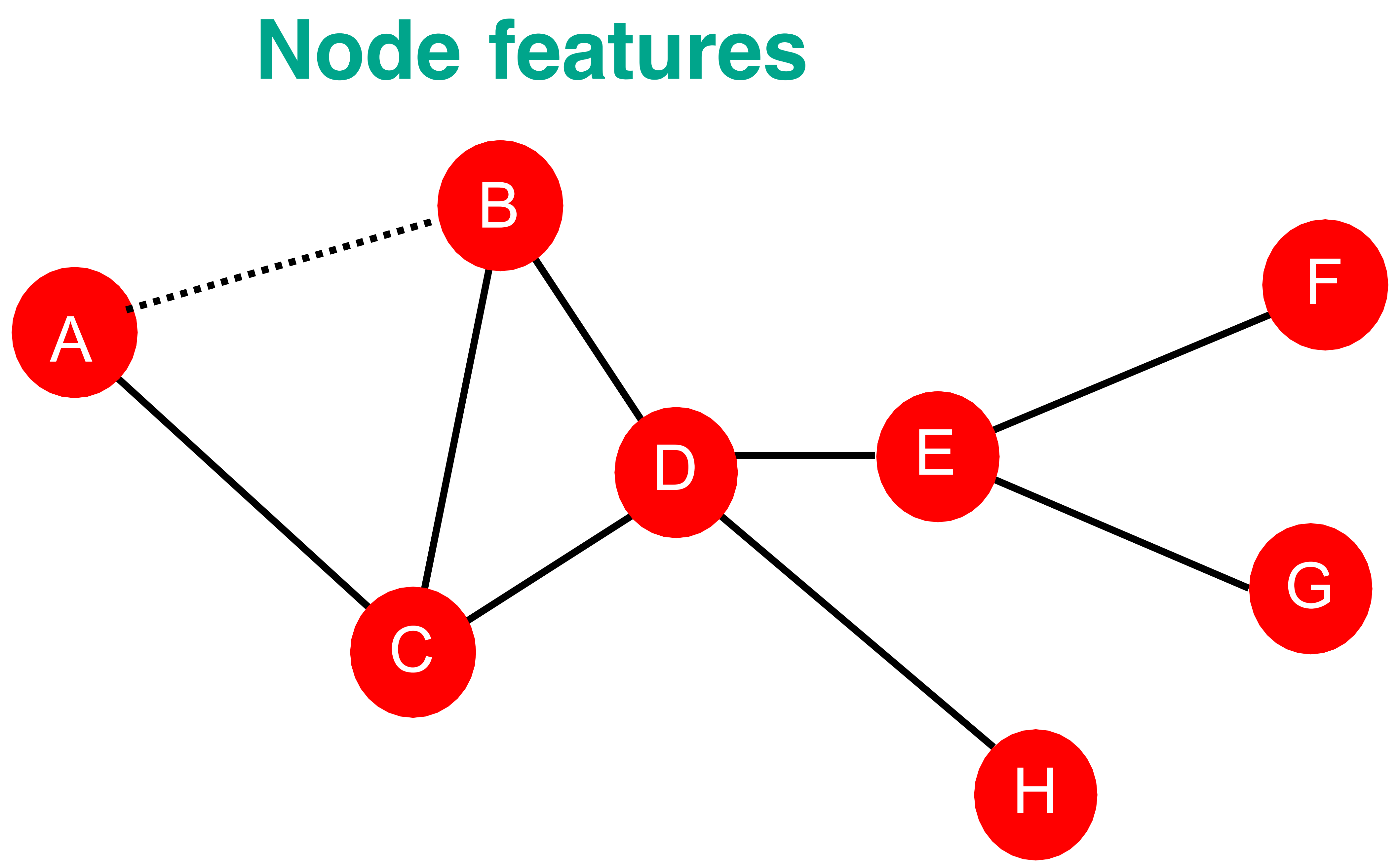
ML needs features.

Node-Level Overview

<u>**Goal**</u>

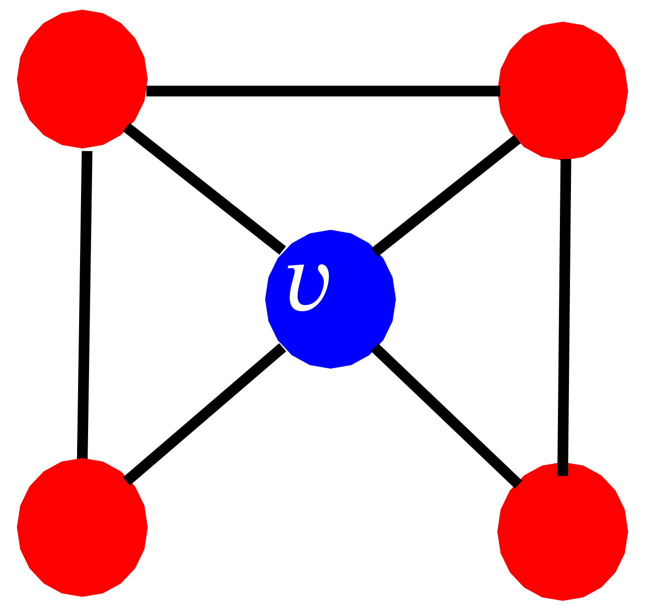
Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- **Graphlets**

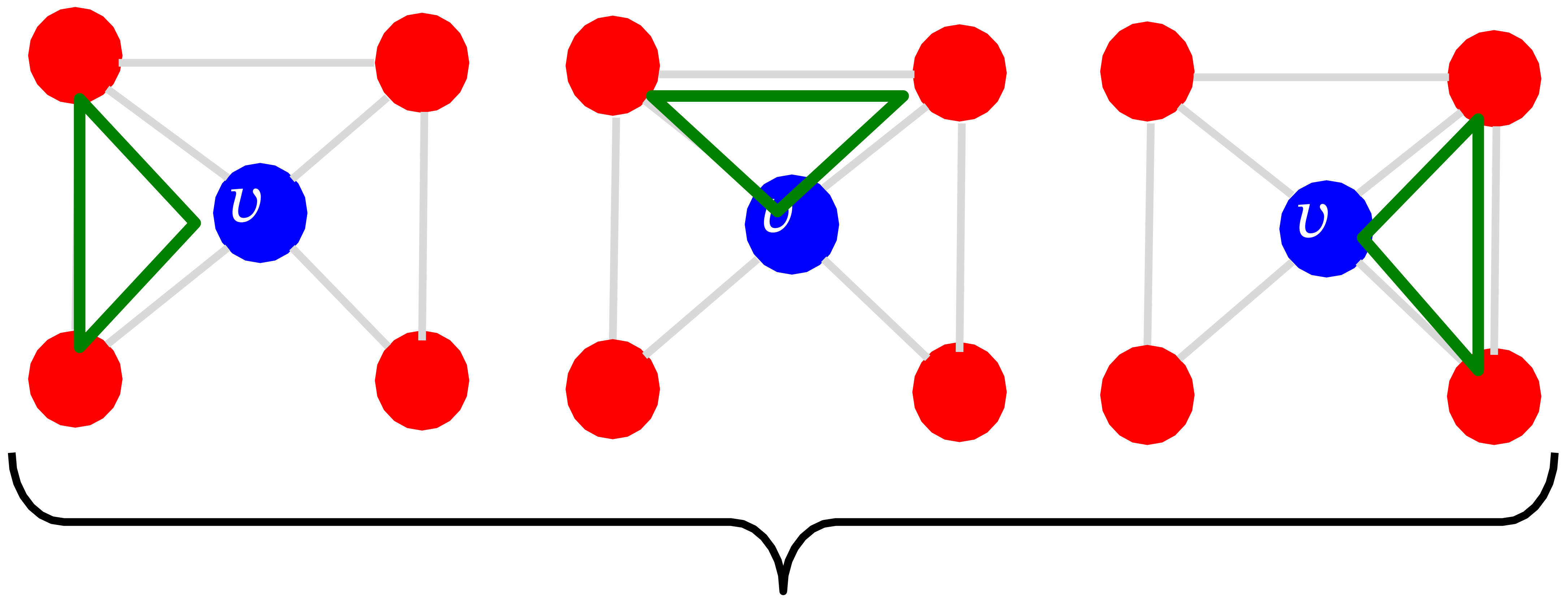


Node Features: Graphlets

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



$e_v = 0.5$



3 triangles (out of 6 node triplets)

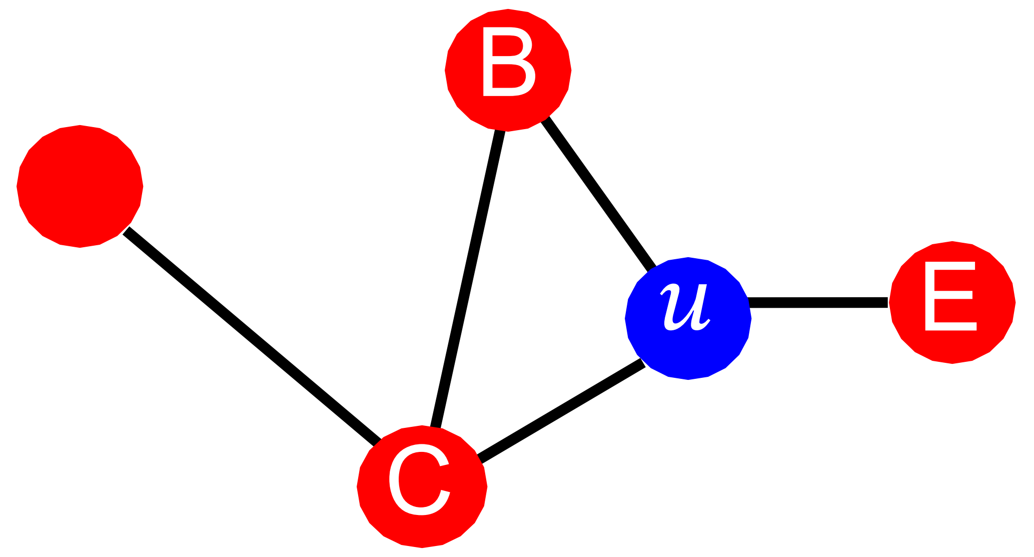
- We can generalize the above by counting **#(pre-specified subgraphs**, i.e., **graphlets)**

Node Features: Graphlets

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

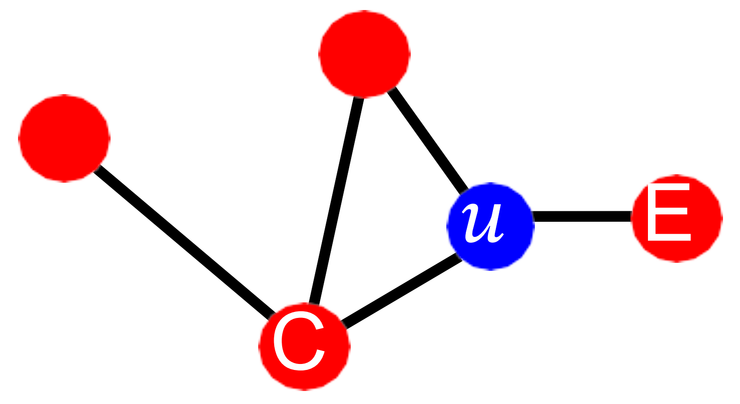
Analogy:

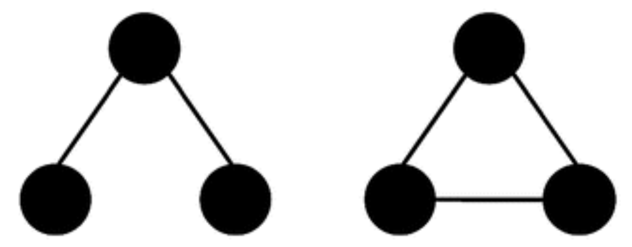
- Degree counts **#(edges)** that a node touches
- Clustering coefficient counts #(of **triangles)** that a node touches.
- <u>**Graphlet Degree Vector (GDV)**</u>: Graphlet-base features for nodes
  - GDV counts **#(graphlets)** that a node touches



Node Features: Graphlets

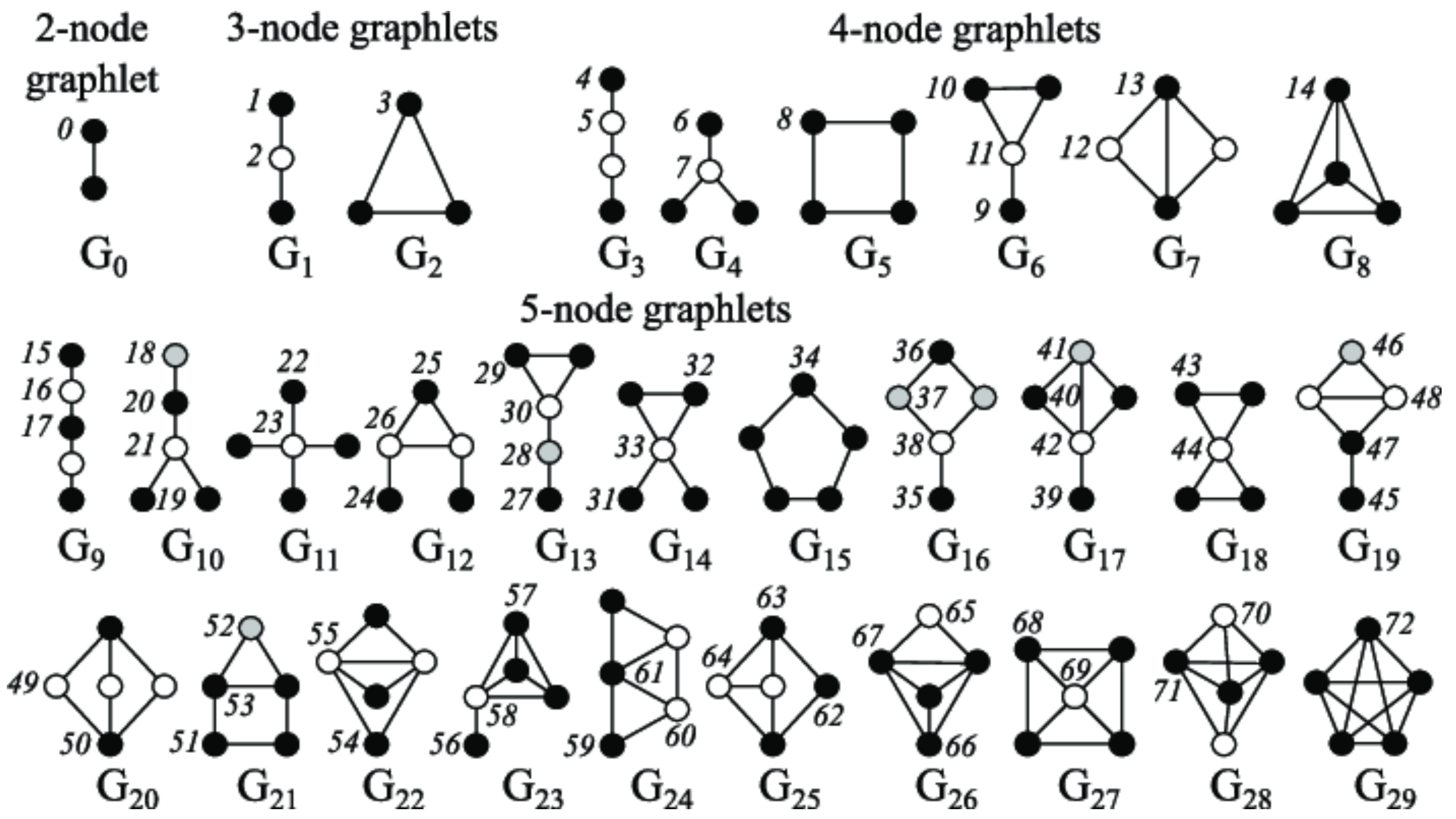
- Considering graphlets of size 2-5 nodes we get:
  - A vector of 73 coordinates is a signature of a node that describes the topology of the node's neighborhood
- Graphlet degree vector provides a measure of a **node's local network topology**:
  - Comparing vectors of two nodes provides a more detailed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





Node Features: Graphlets

Rooted connected induced non-isomorphic subgraphs



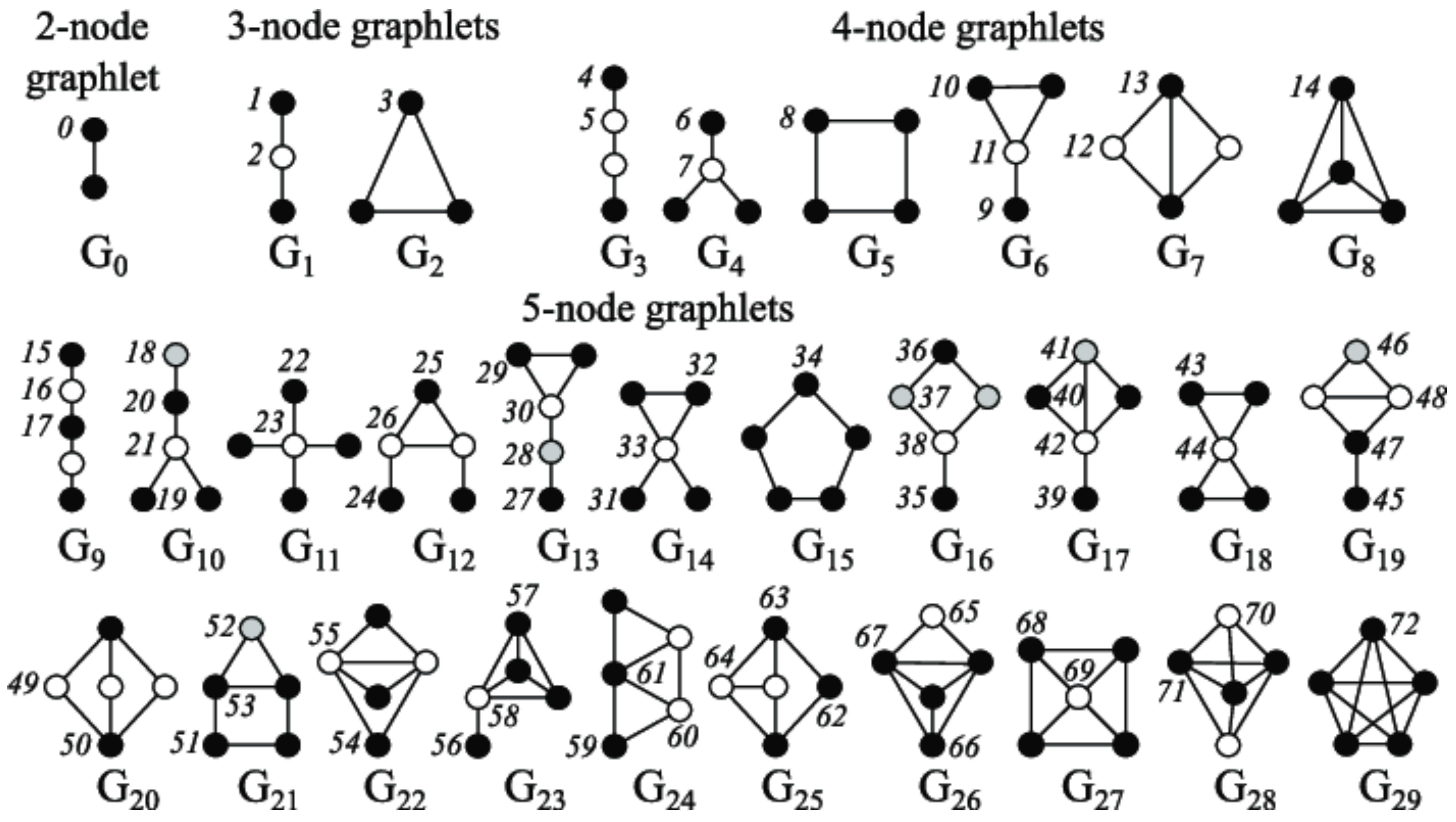
There are 30 different graphlets on up to 5 nodes

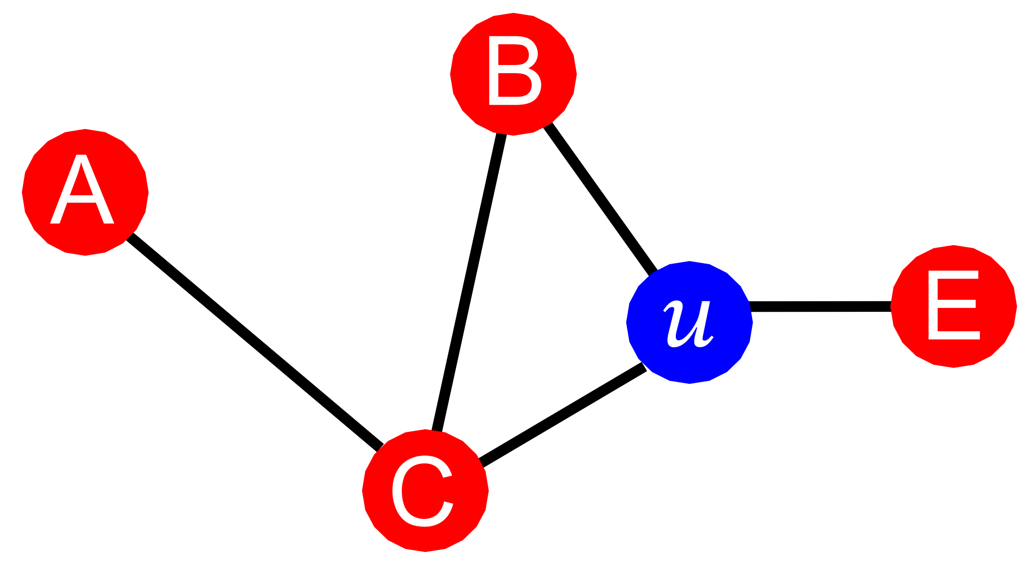
There are 73 distinct automorphism orbits (or simply orbits) for graphlets with up to 5 nodes.

Orbits are the distinct roles nodes play *within* those graphlets based on symmetry.

Node Features: Graphlets

Rooted connected induced non-isomorphic subgraphs





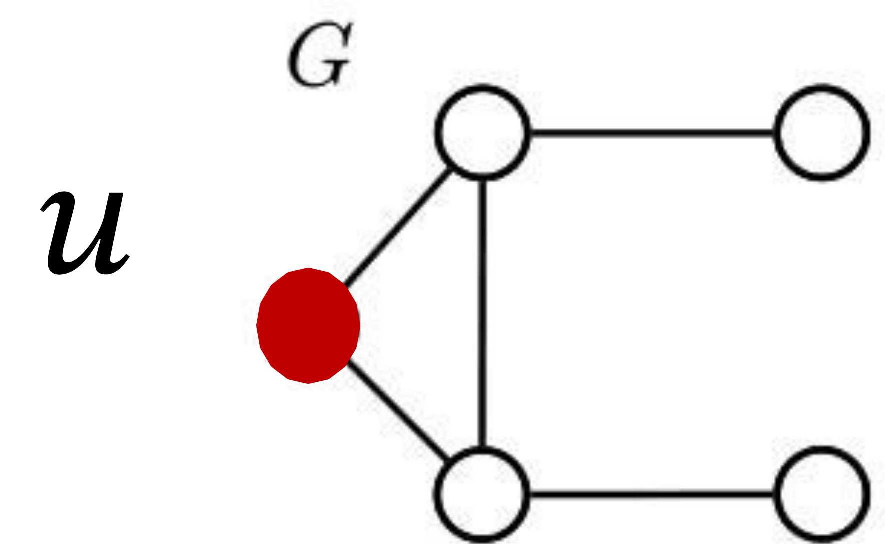
<u>u **has graphlets:**</u>

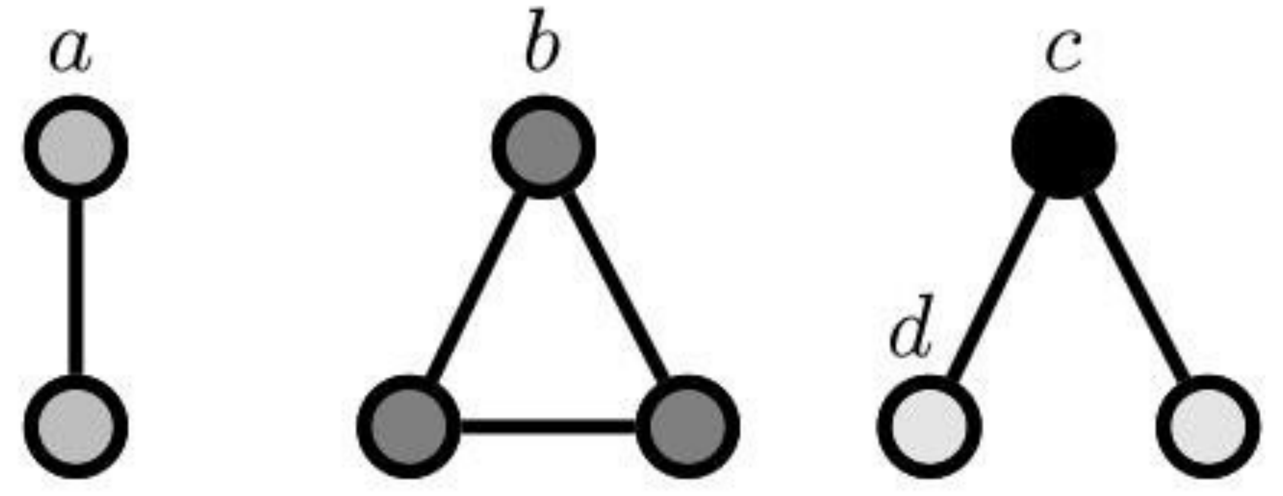
0, 1, 2, 3, 5, 10, 11, …

Node Features: Graphlets

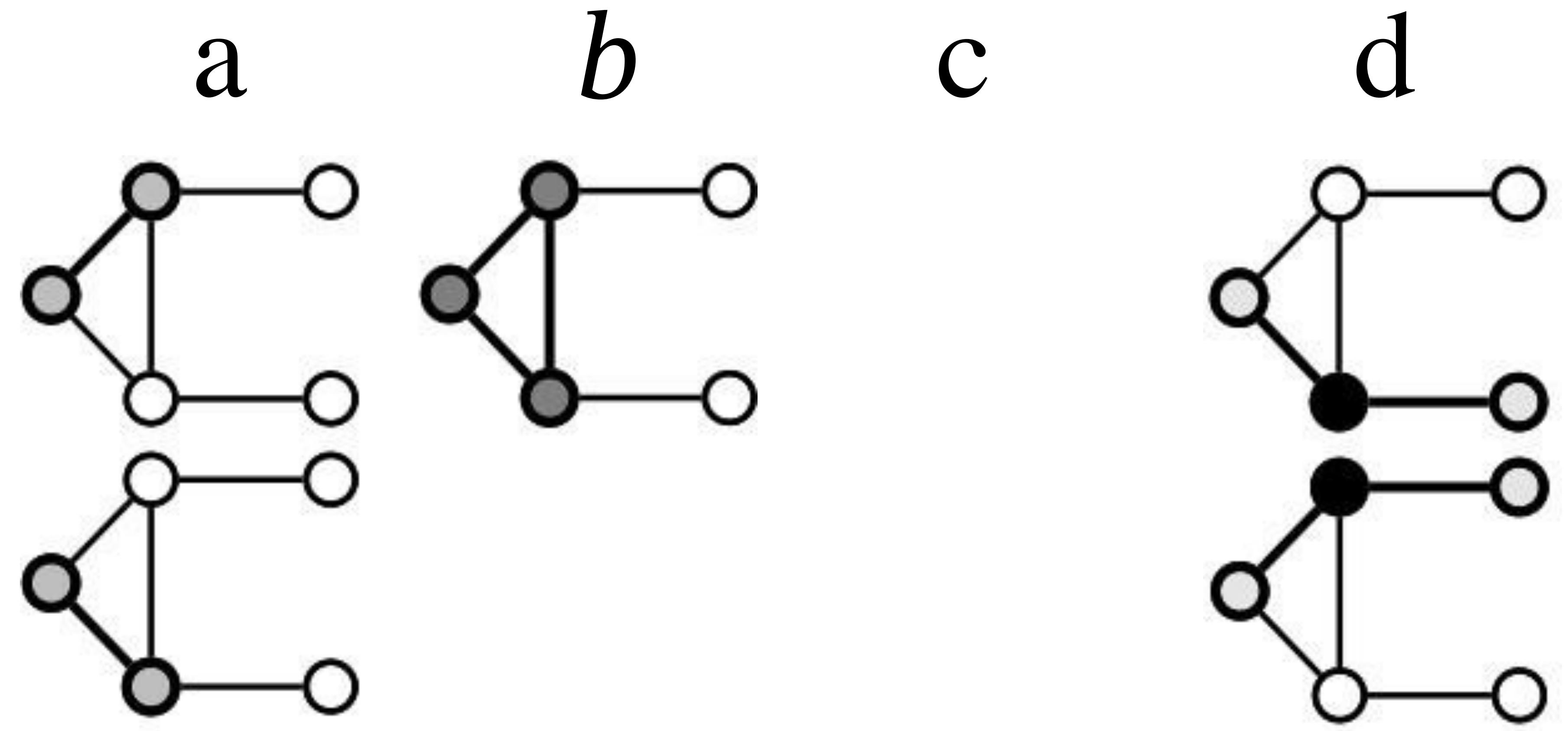
- **Graphlet Degree Vector (GDV)**: A count vector of graphlets rooted at a given node.
- **Example:**

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

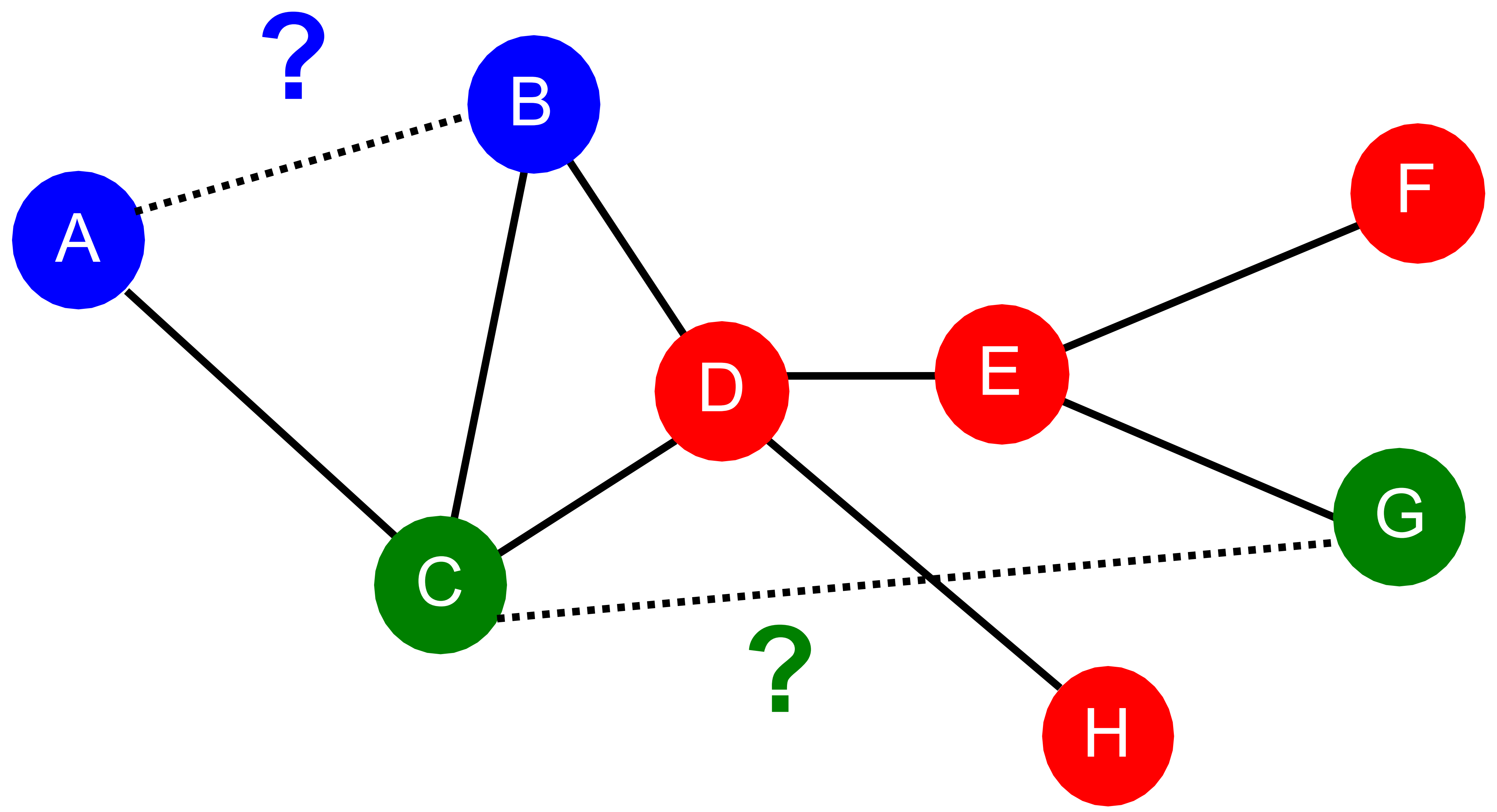
- **Importance-based** features: capture the **importance** of a node in a graph
  - **Node degree**:
    - Counts the number of neighboring nodes
  - **Node centrality**:
    - Models the importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness 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

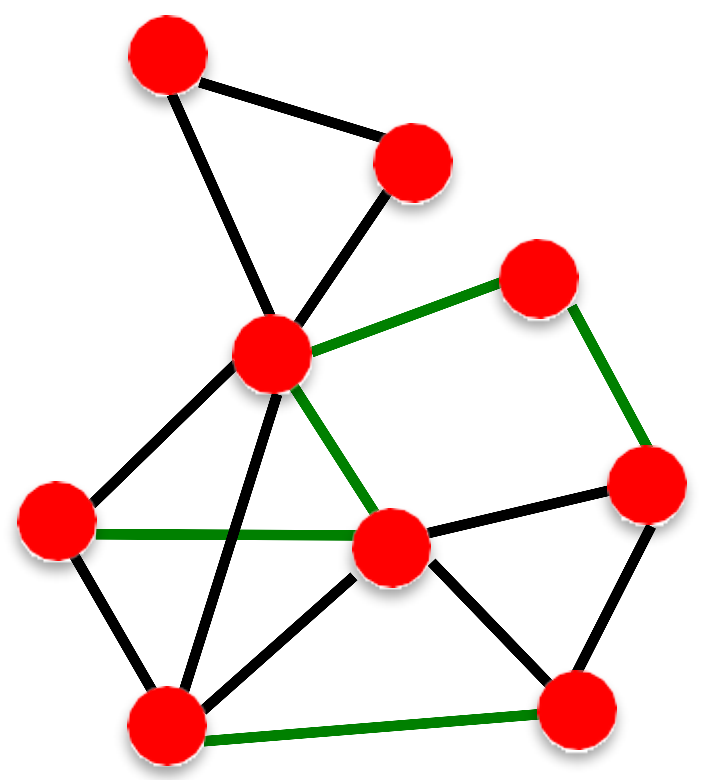
- The task is to **predict new links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for a pair of nodes.



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

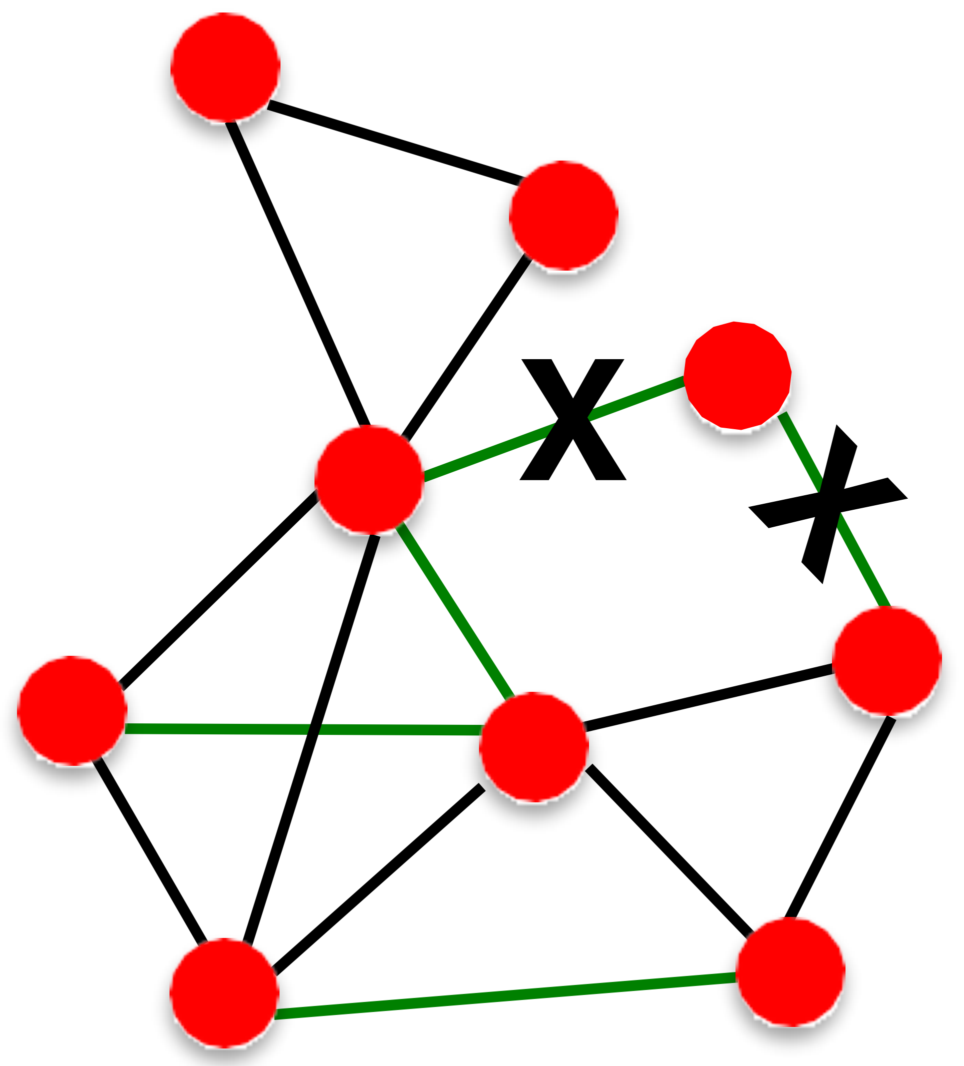


$G[t_0, t_0']$

$G[t_1, t_1']$

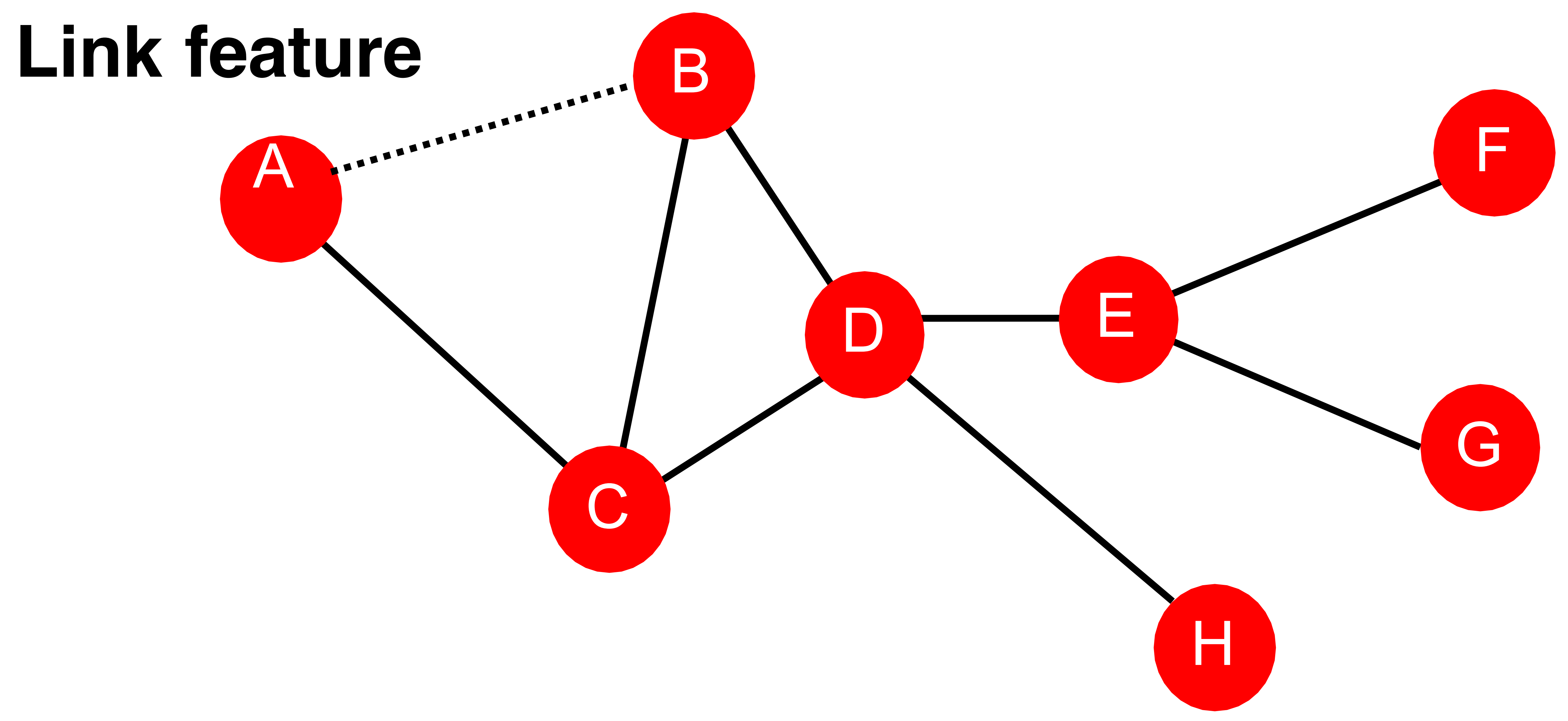
Link Prediction via Proximity

- **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 $G[t_1, t_1']$



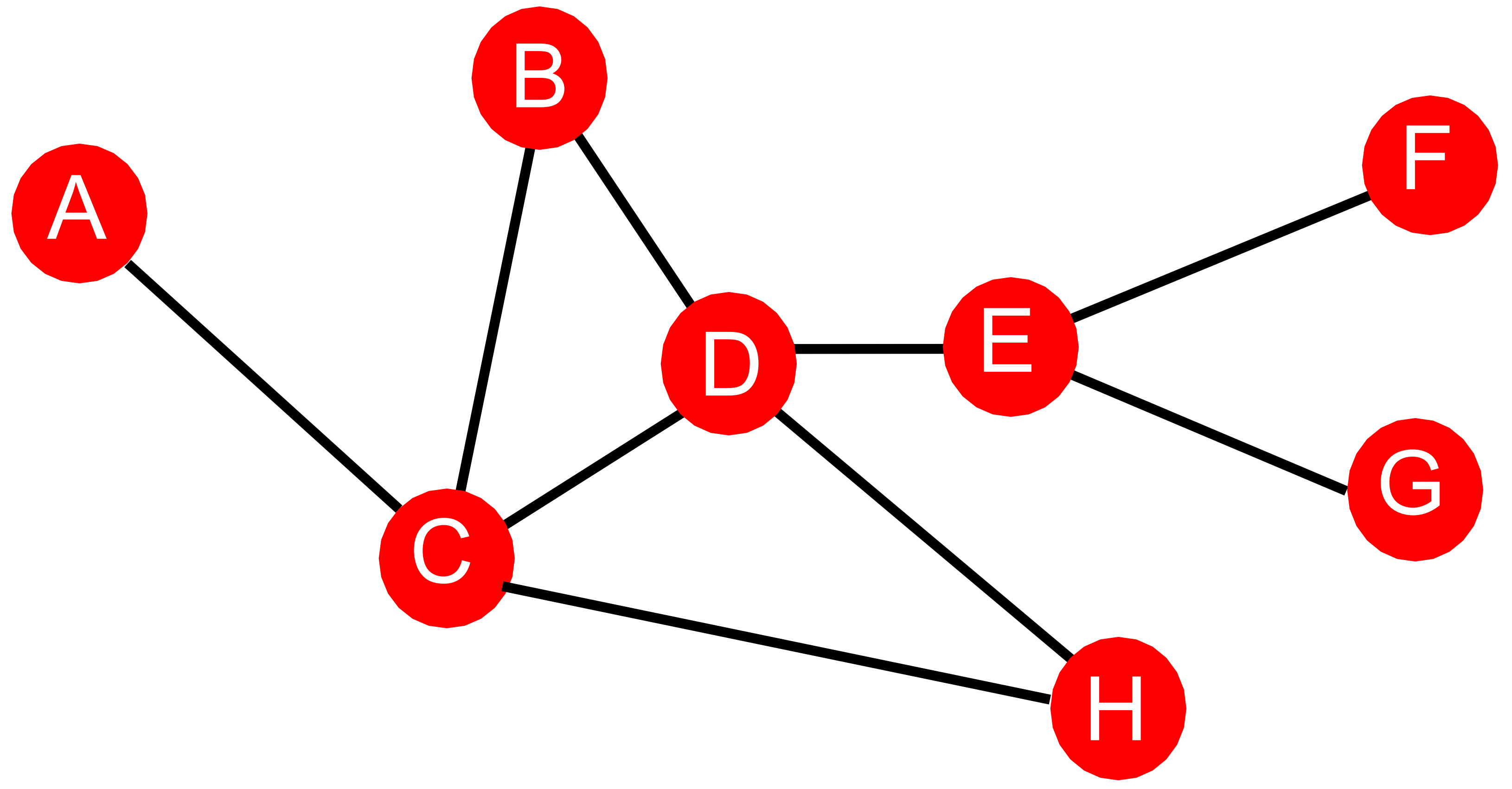
Link-Level Features: Overview

- **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

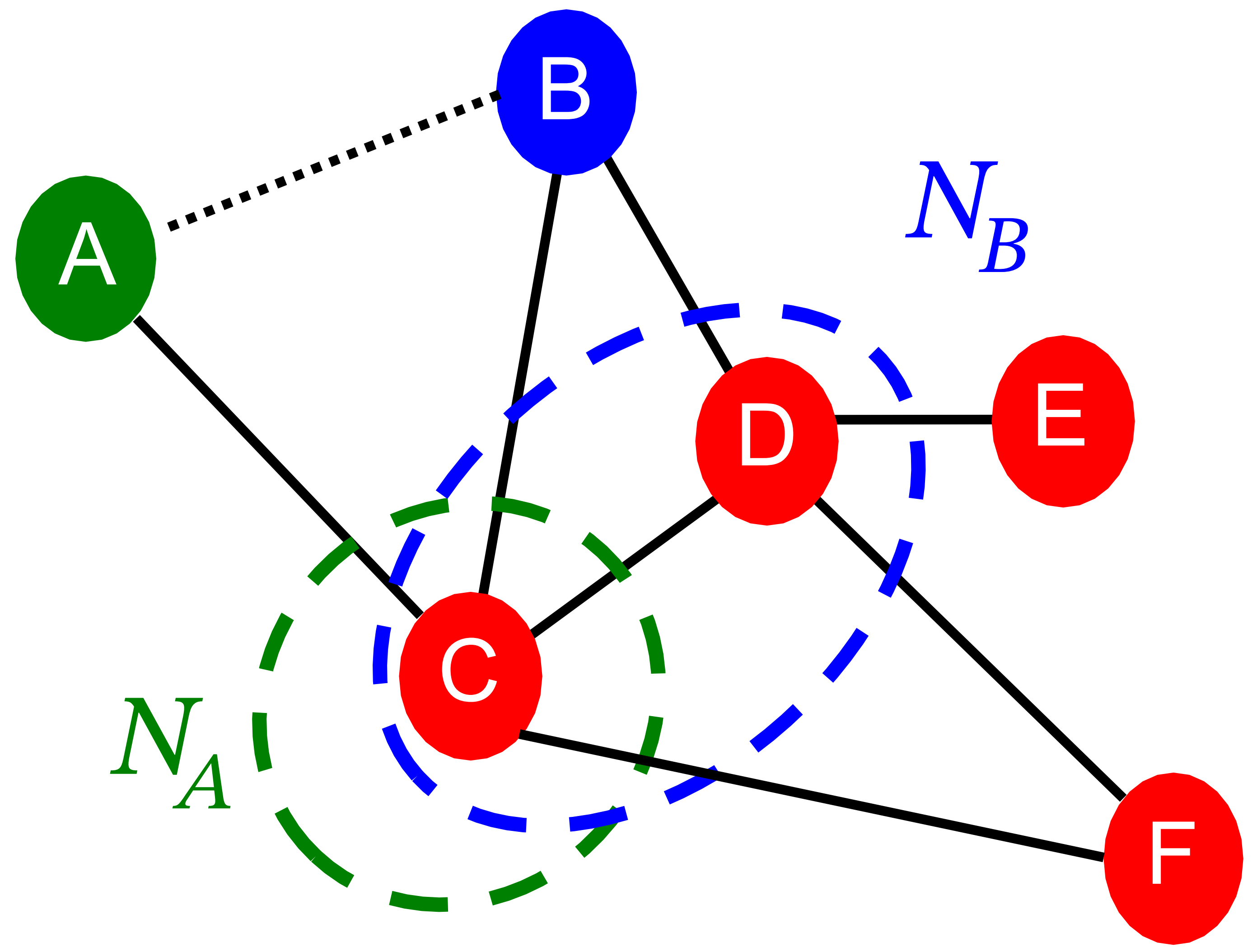
Captures # neighboring nodes shared between two nodes v_1 and v_2

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

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



Key differences

Common Neighbors:

- Simple absolute count of shared neighbors.
- Higher count = higher similarity, but doesn't consider node size/graph density.

Jaccard Index:

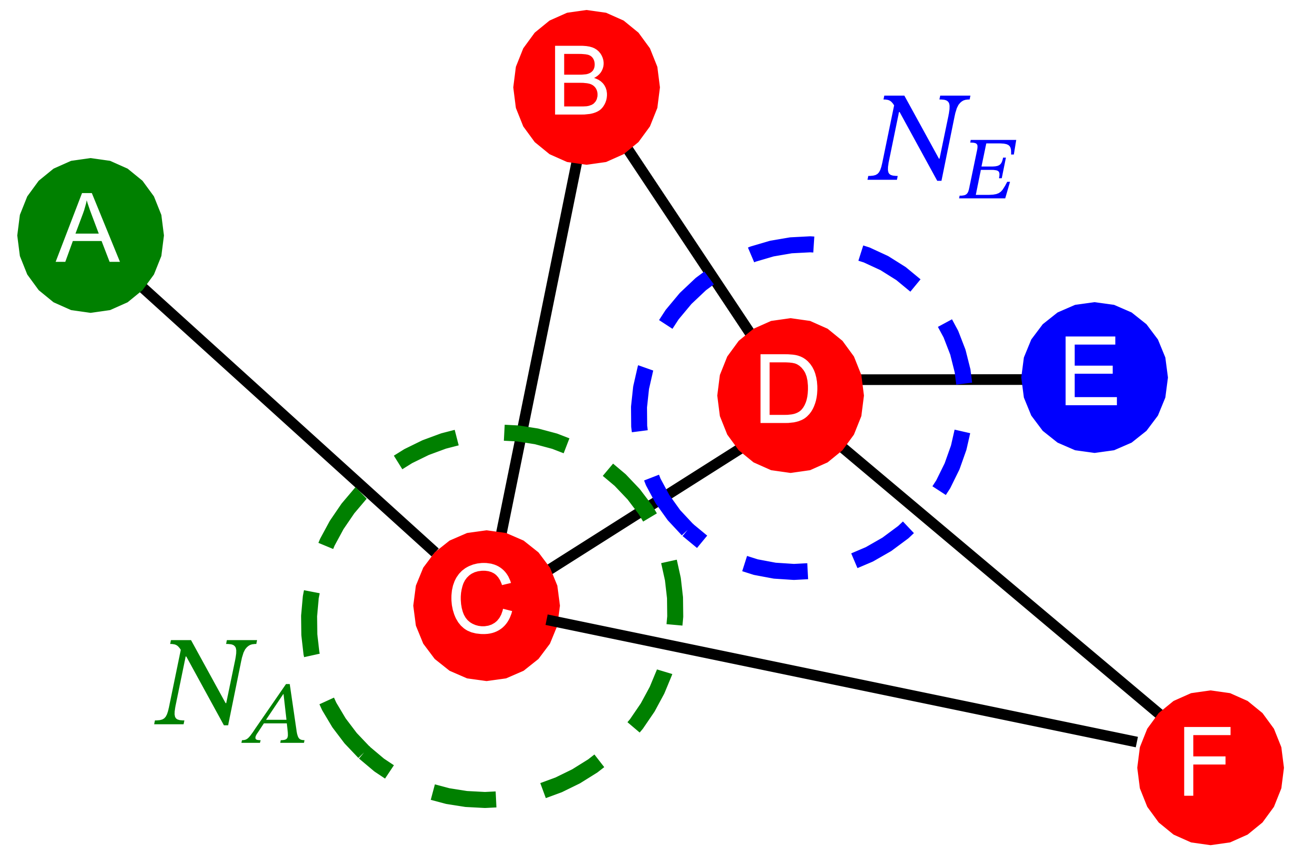
- Normalizes shared count by the total unique neighbors.
- Gives a relative proportion (0 to 1), better for comparing nodes of different sizes.

Adamic-Adar Index:

- Weights shared neighbors by the inverse of their degree.
- More weight given to common neighbors who are less connected themselves (rare).

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.**



$$N_A \cap N_E = \phi$$

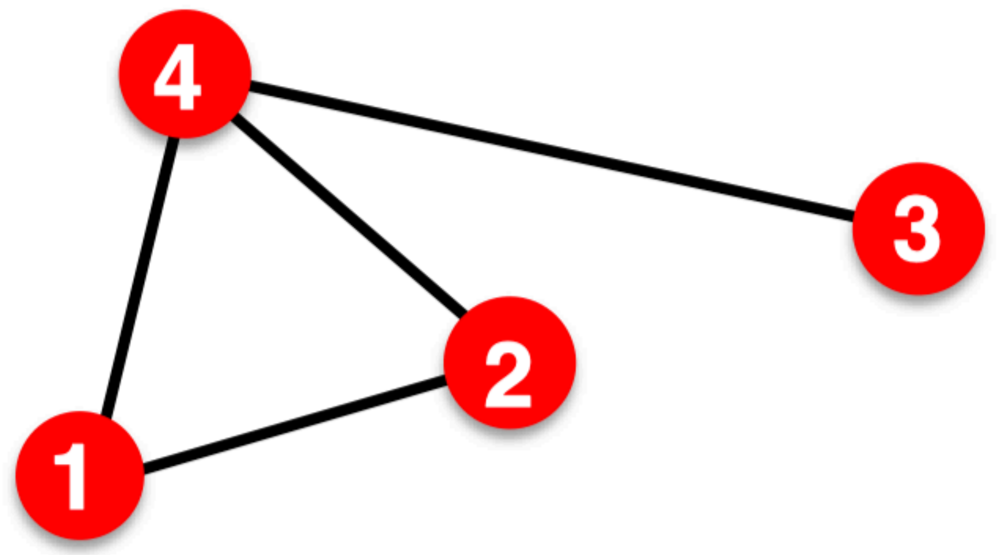
$$|N_A \cap N_E| = 0$$

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 $P_{uv}^{(K)}$ = **#walks of length K between u and v**
  - **We will show** $P^{(K)} = A^k$
  - $P_{uv}^{(1)}$ = #walks of length 1 (direct neighborhood) between u and v = A_{uv}



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

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

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**

$$A^2 = \begin{pmatrix} 0 & \boxed{1} & 0 & \boxed{1} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & \boxed{0} & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & \boxed{1} & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & \boxed{1} & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Power of adjacency

Global Neighborhood Overlap

- **Katz index: <u>count the number of walks of all lengths between a pair of nodes.</u>**
- 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:

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

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

- Katz index matrix is computed in closed form:

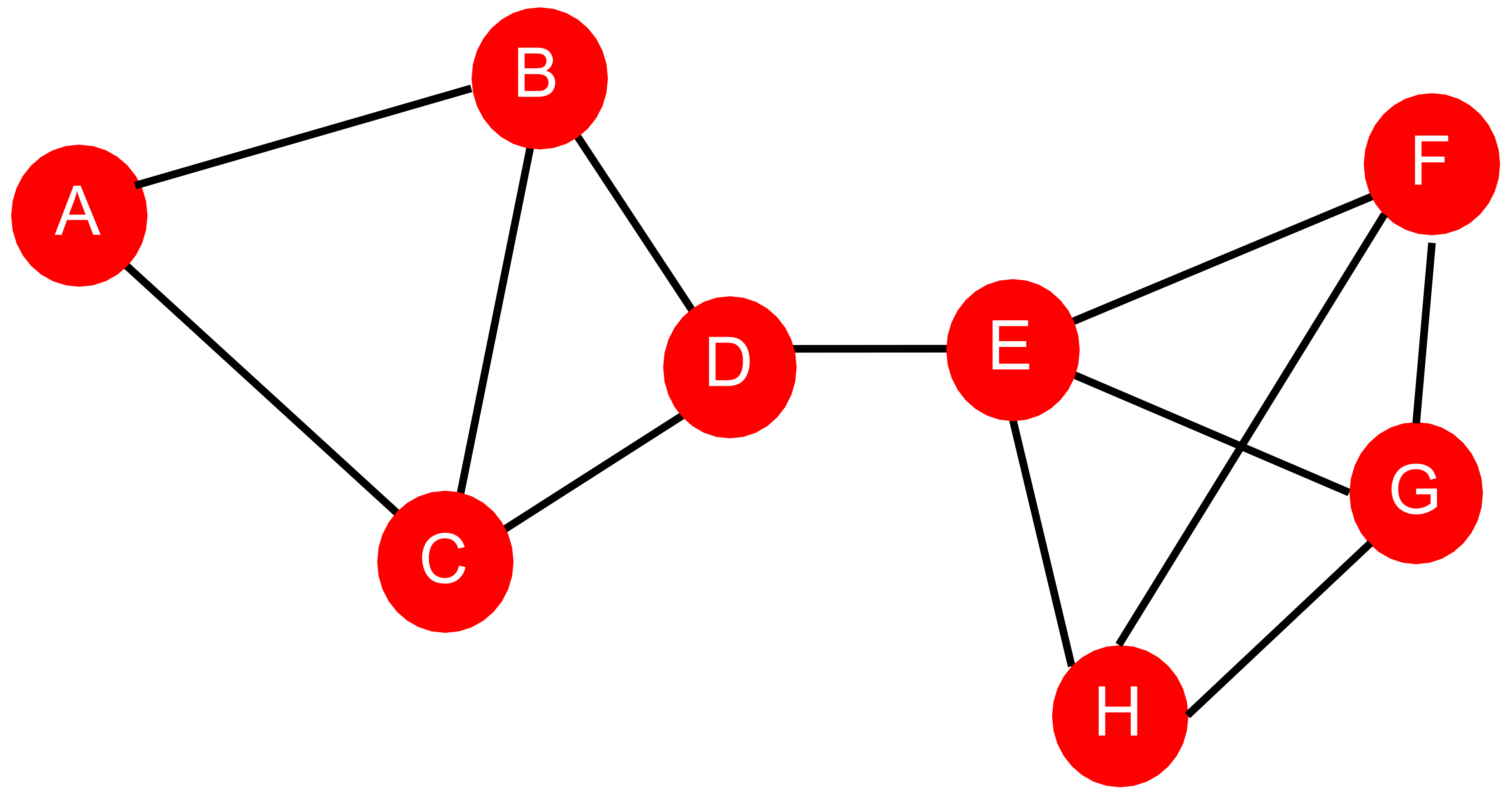
$S = \sum_{i=1}^{\infty} \beta^i A^i = (I - \beta A)^{-1} - I$ by geometric series of matrices

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

- Goal: We want to use machine learning (ML) to make predictions about entire graphs (e.g., classify molecules, detect communities).
  - Standard ML Approach: ML models usually work with feature vectors - lists of numbers describing each data point.
  - The Problem with Graphs: How do you represent a complex graph as a simple list of numbers (a feature vector)?
  - Designing good, informative features by hand that capture the graph's structure can be very difficult and time-consuming.
- **The Kernel Idea:** What if, instead of defining features first, we directly define a way to measure similarity between two graphs? This similarity measure is called a **kernel**.

Graph-Level Features: Overview

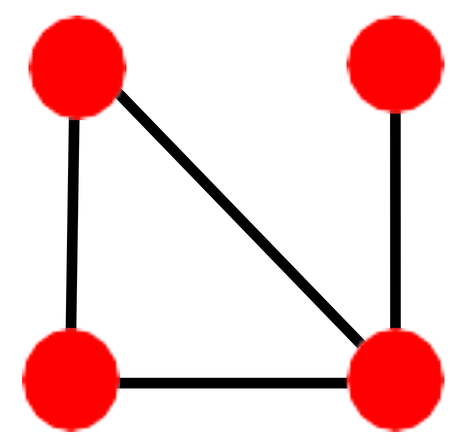
Graph Kernels:

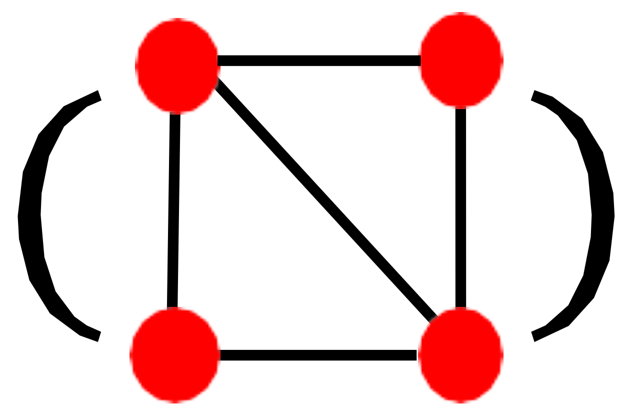
- Measure the similarity between two graphs:
  - Graphlet Kernel [1]
  - Weisfeiler-Lehman Kernel [2]
  - Other kernels are also proposed in the literature
  - (beyond the scope of this lecture)
    - Random-walk kernel
    - Shortest-path graph kernel
    - And many more…

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…



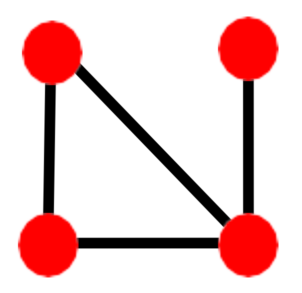
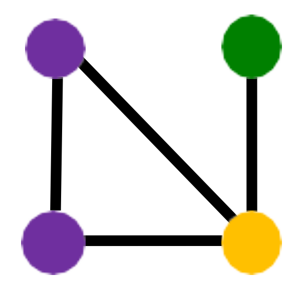


Graph Kernel: Key Idea

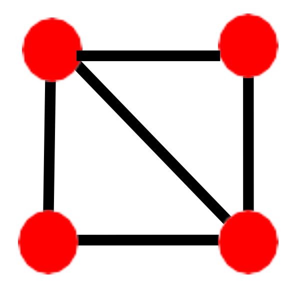
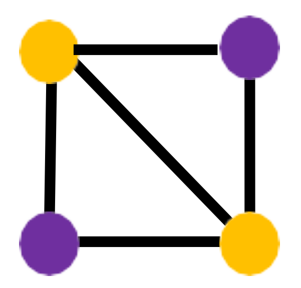
What if we use **Bag of node degrees**?

Deg1: ● Deg2: ● Deg3: ●

● ● ●

$\phi($  $)$ = count() = [1, 2, 1]

Obtains different features for different graphs!

$\phi($  $)$ = count() = [0, 2, 2]

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

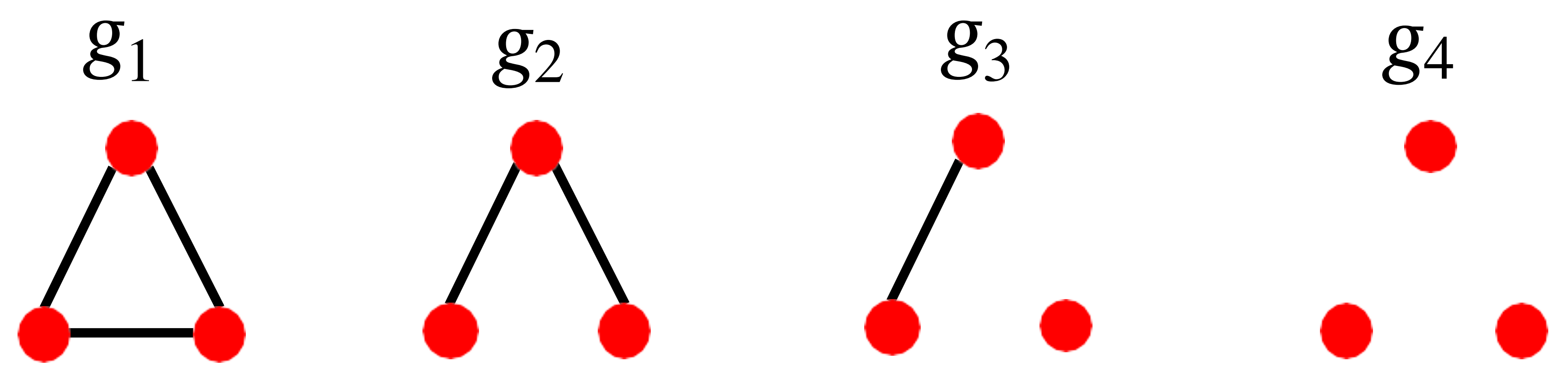
Graph-Level Graphlet Features

- **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.

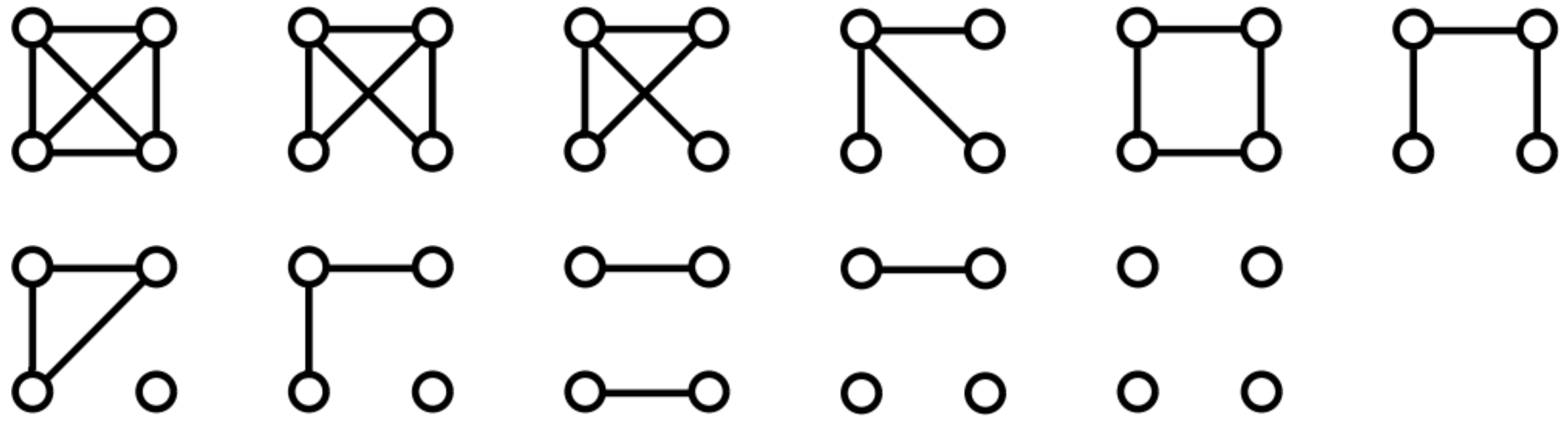
Graph-Level Graphlet Features

Let $\mathcal{G}k = (g1, g2, \ldots, g_{n_k})$ be a list of graphlets of size k.

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



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



Shervashidze *et al.*, AISTATS 2011

Graph-Level Graphlet Features

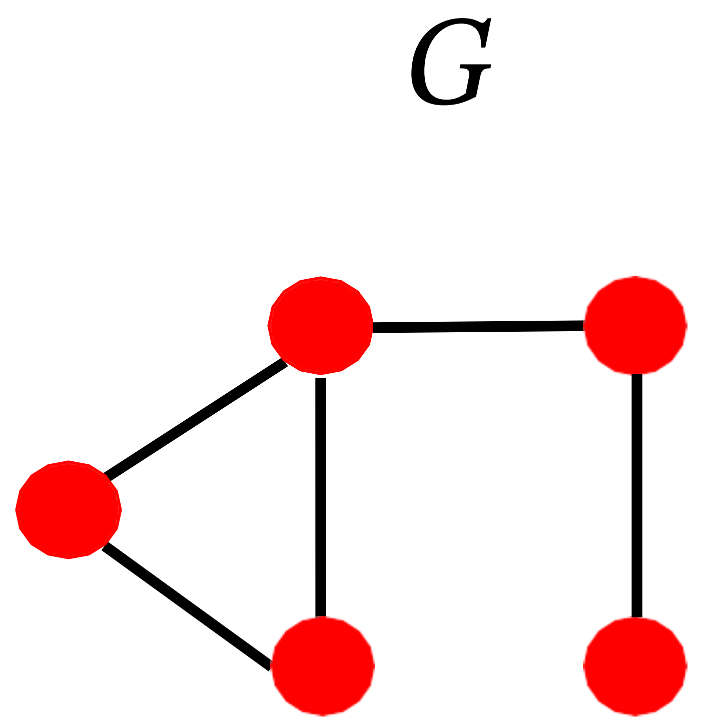
Given graph G and a graphlet list $G_k = (g_1, g_2, \ldots, g_{n_k})$

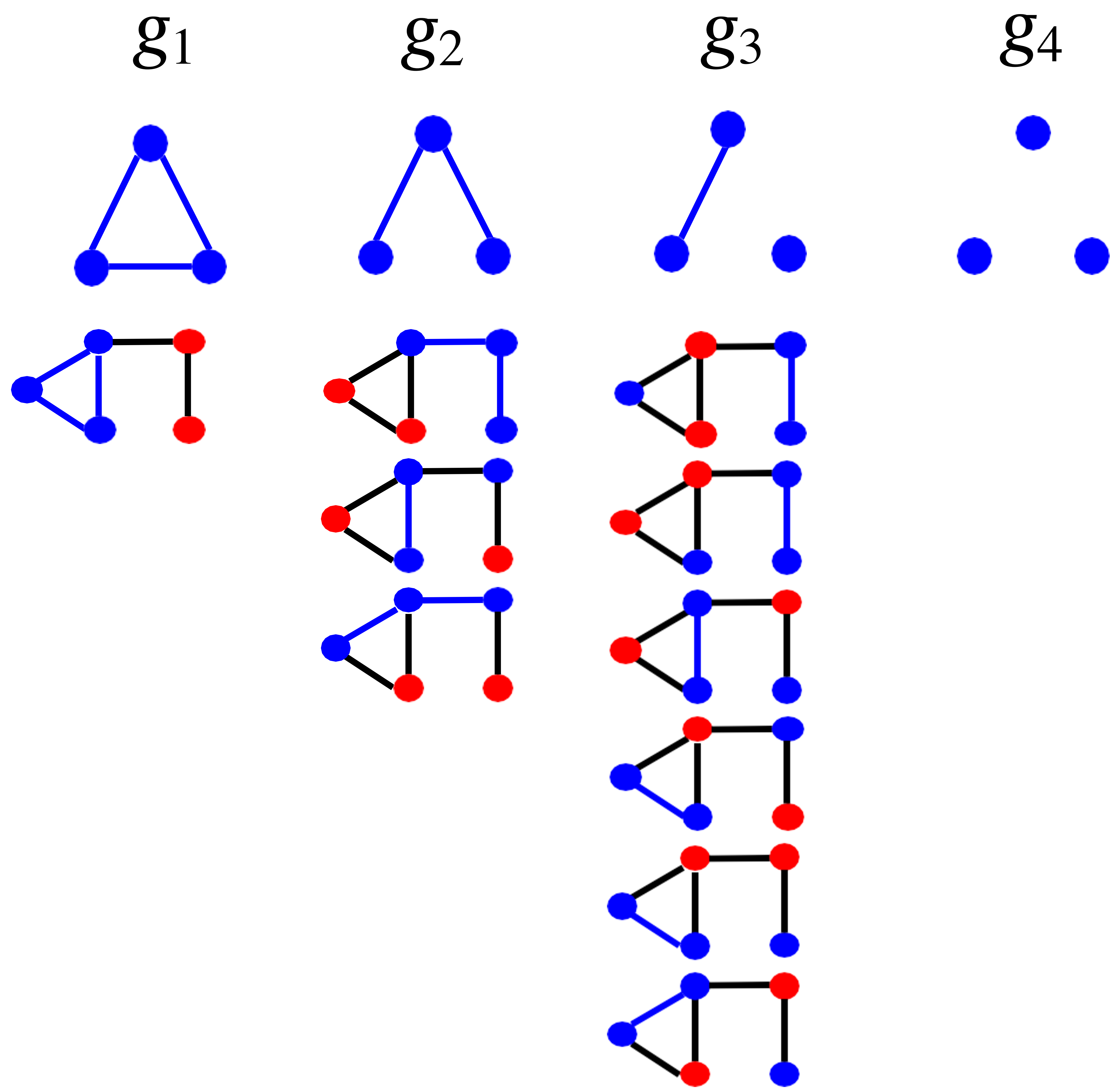
define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

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

Graph-Level Graphlet Features

Example for $k = 3$.





$$\boldsymbol{f}_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$$

Graph-Level Graphlet Kernel

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

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

Problem: if G and G' have different sizes, that will greatly skew the value.
Solution: normalize each feature vector

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

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

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

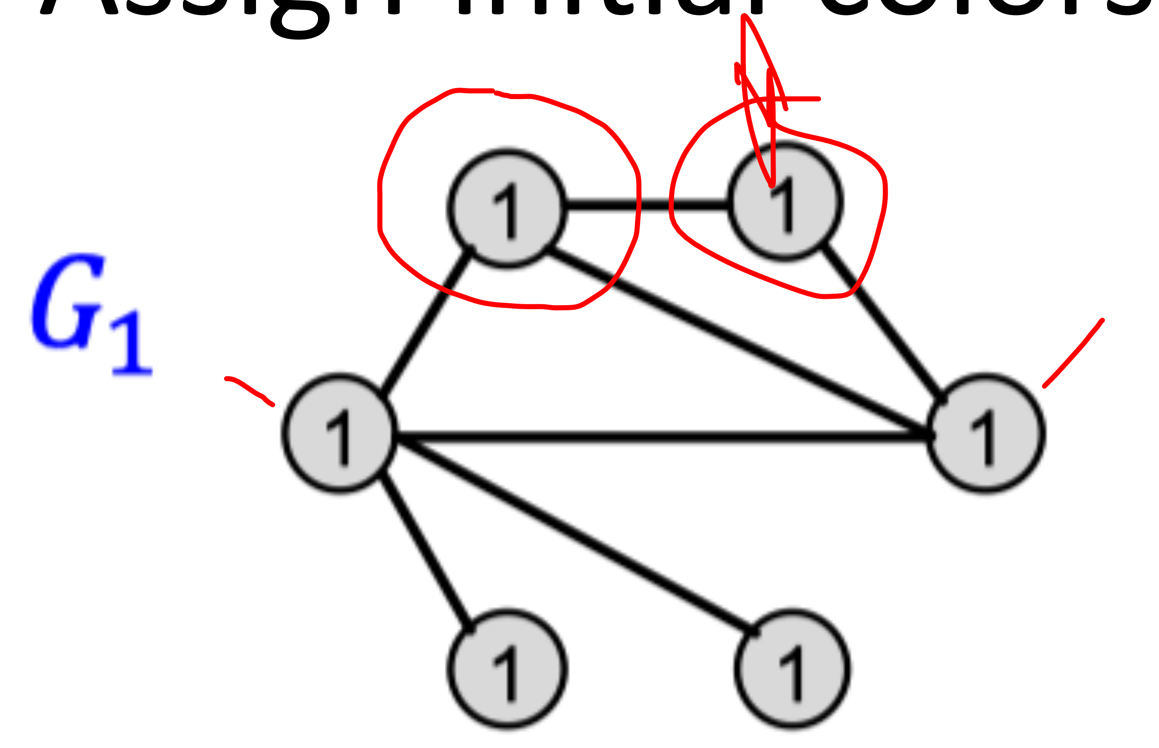
where HASH maps different inputs to different colors

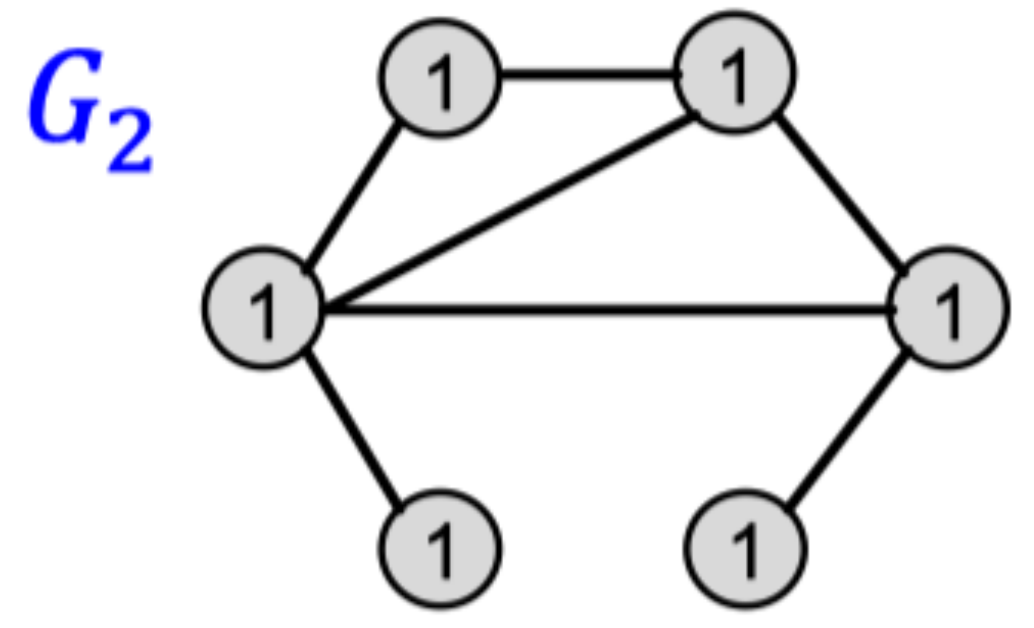
- After k steps of color refinement, $c^{(K)}(v)$ summarizes the structure of the K-hop neighborhood

Color Refinement (1)

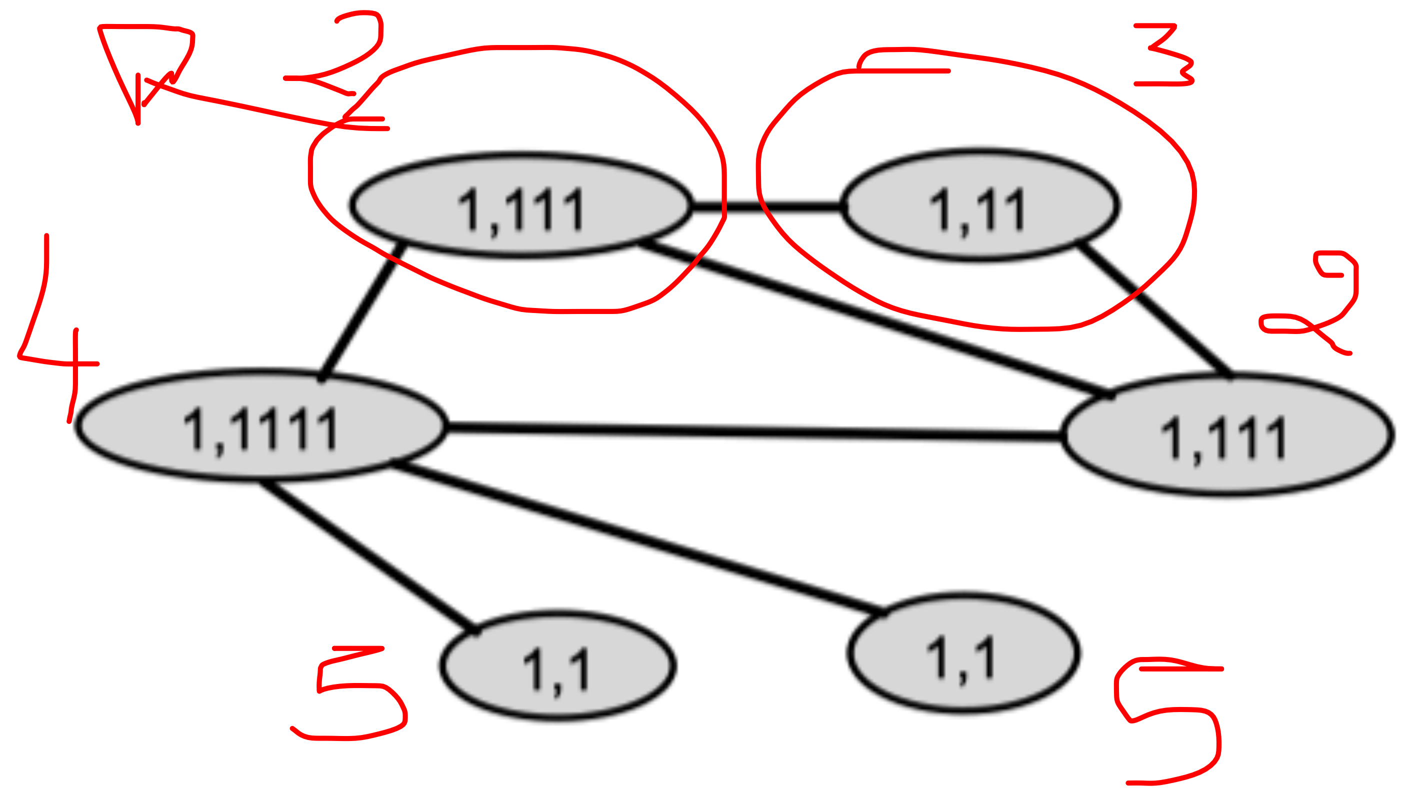
Example of color refinement given two graphs

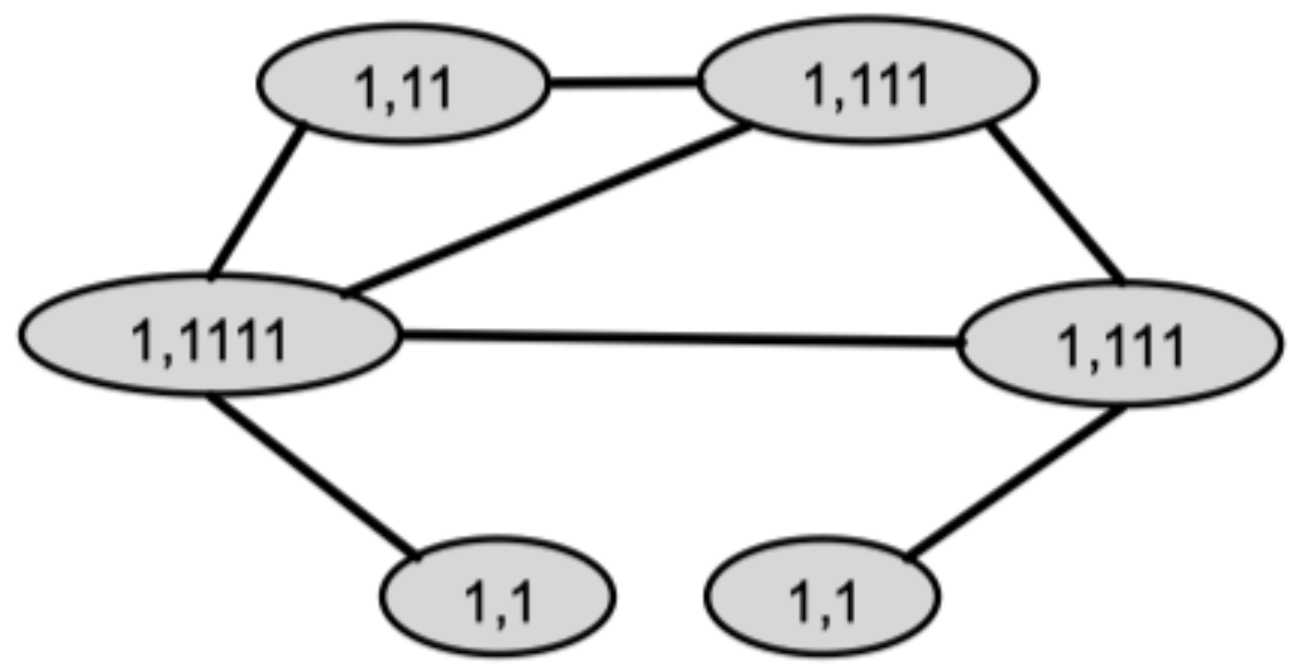
- Assign initial colors





- Aggregate neighboring colors

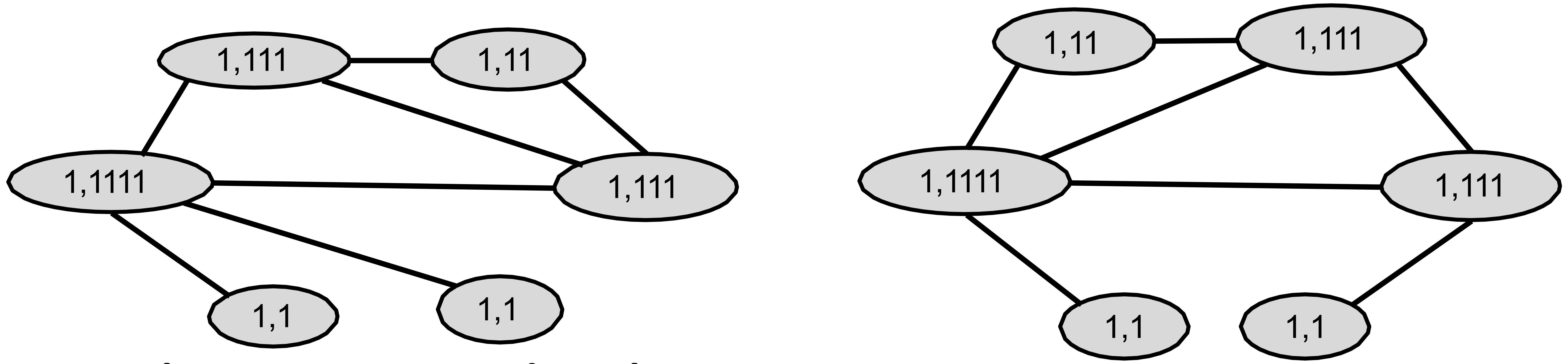




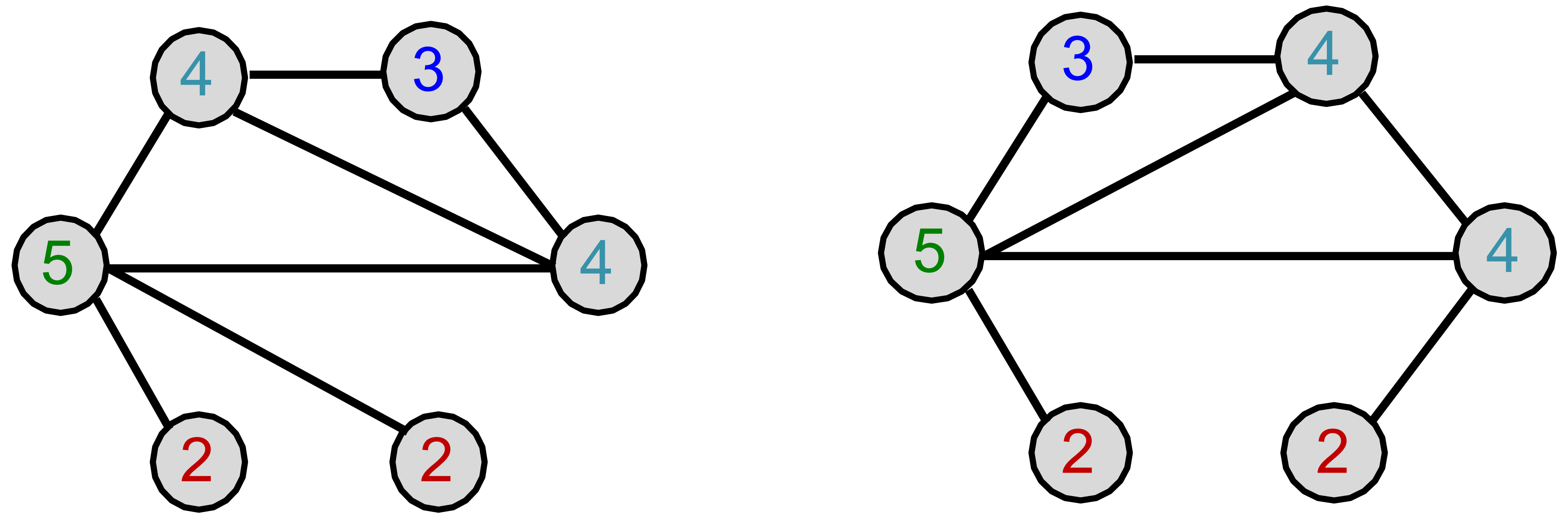
Color Refinement (2)

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors



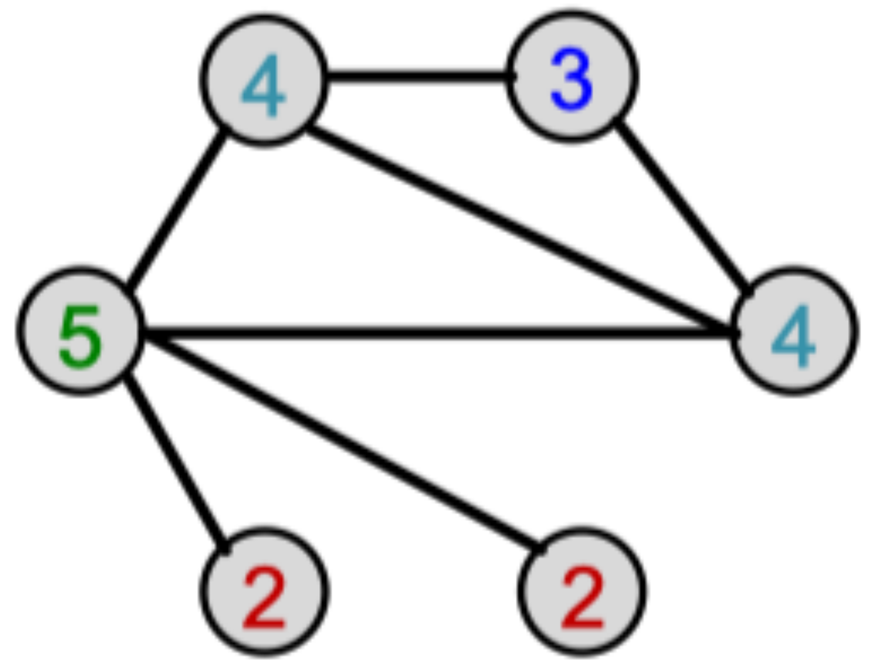
Hash table

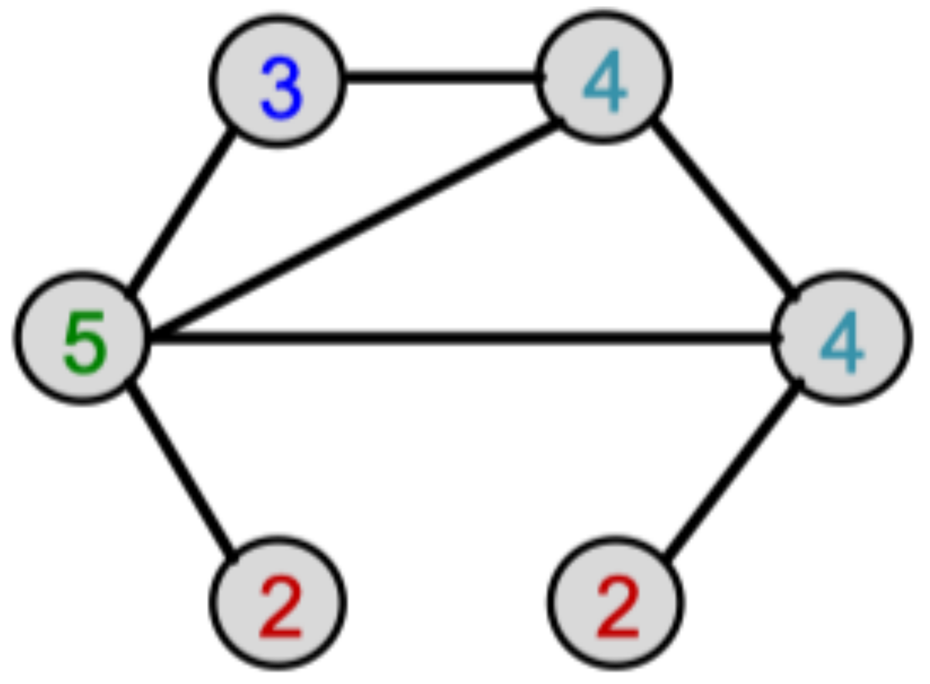
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color Refinement (3)

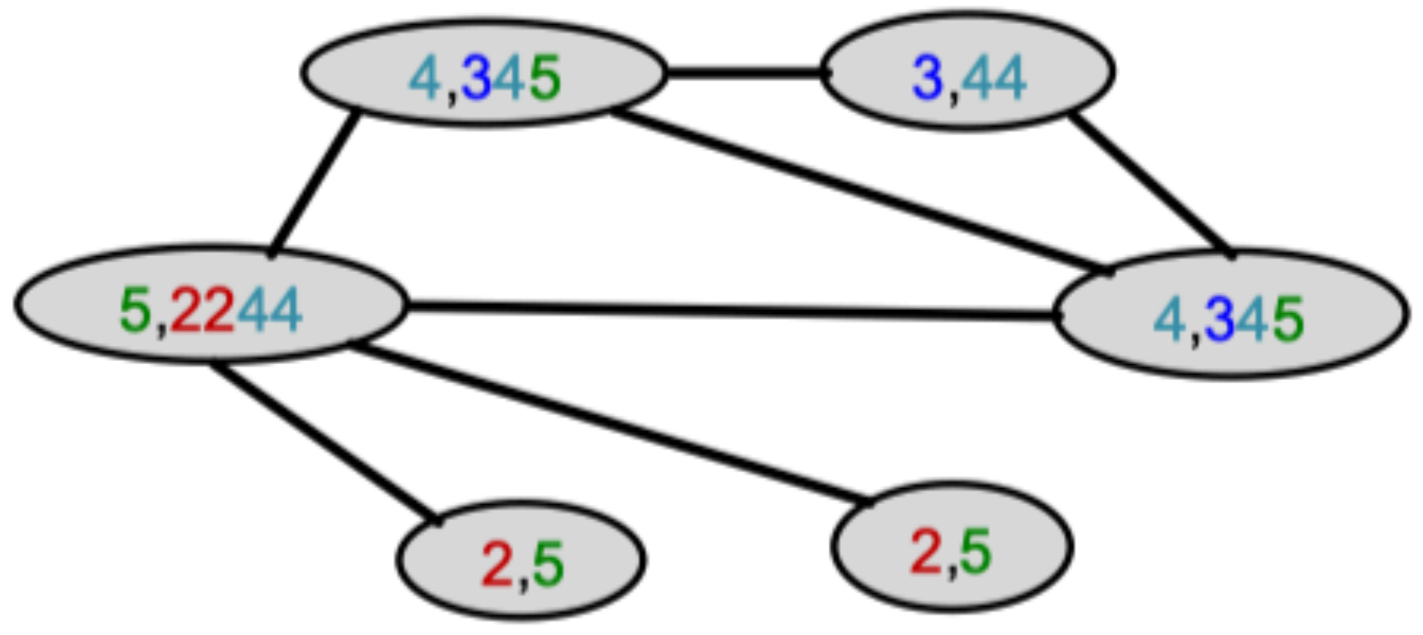
Example of color refinement given two graphs

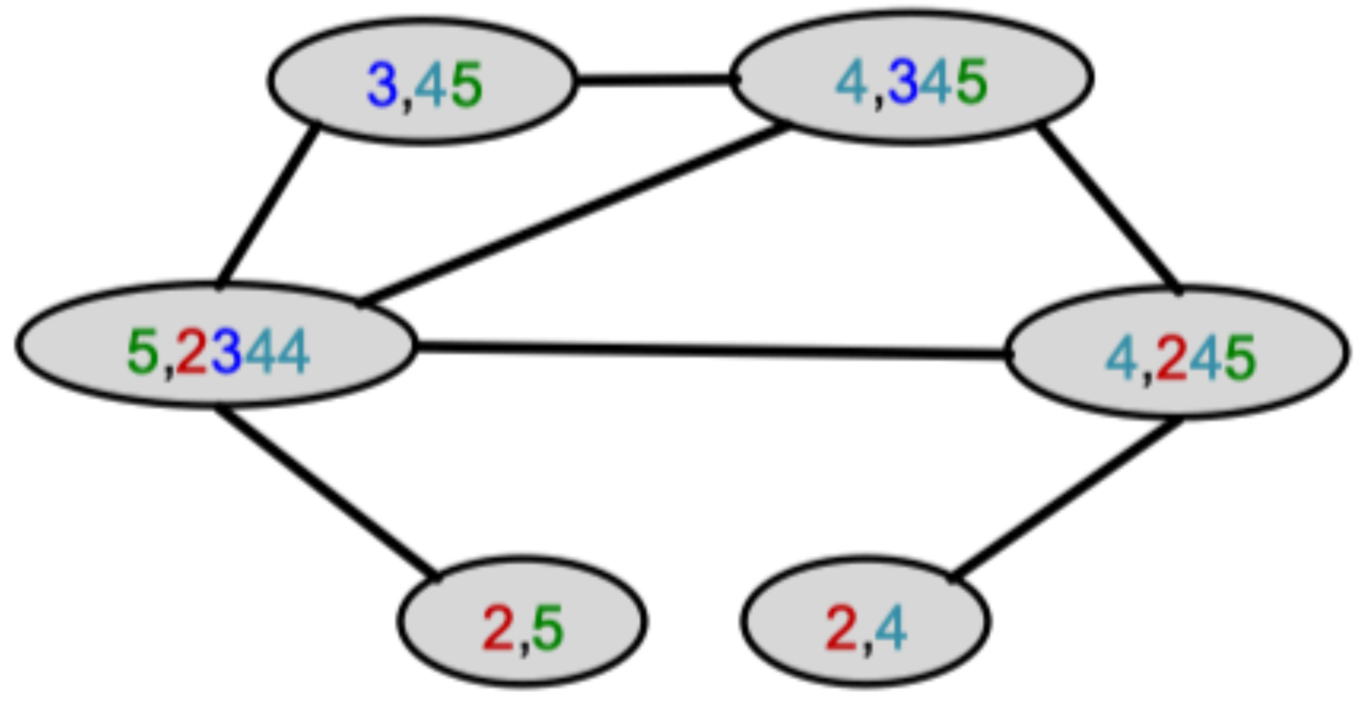
- Aggregated colors





- Hash aggregated colors

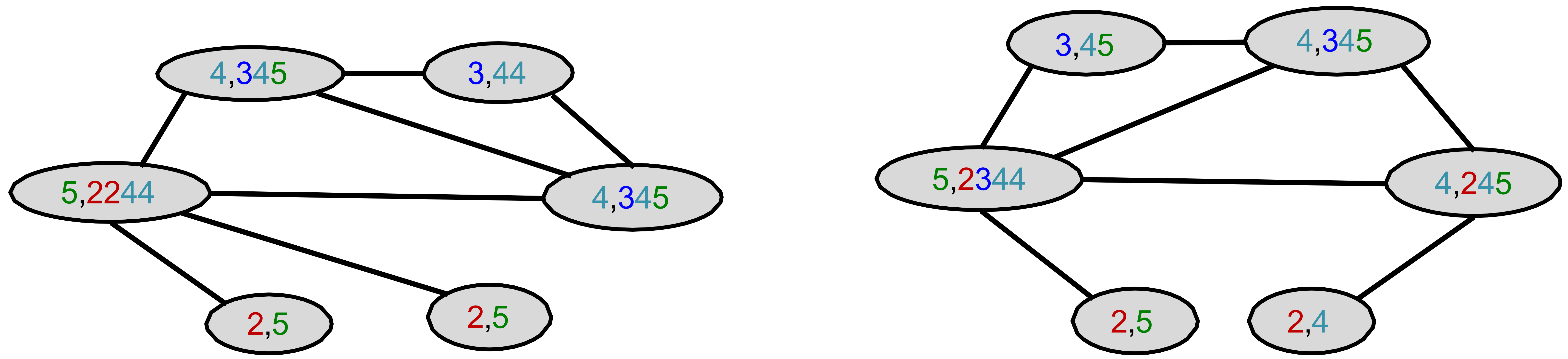




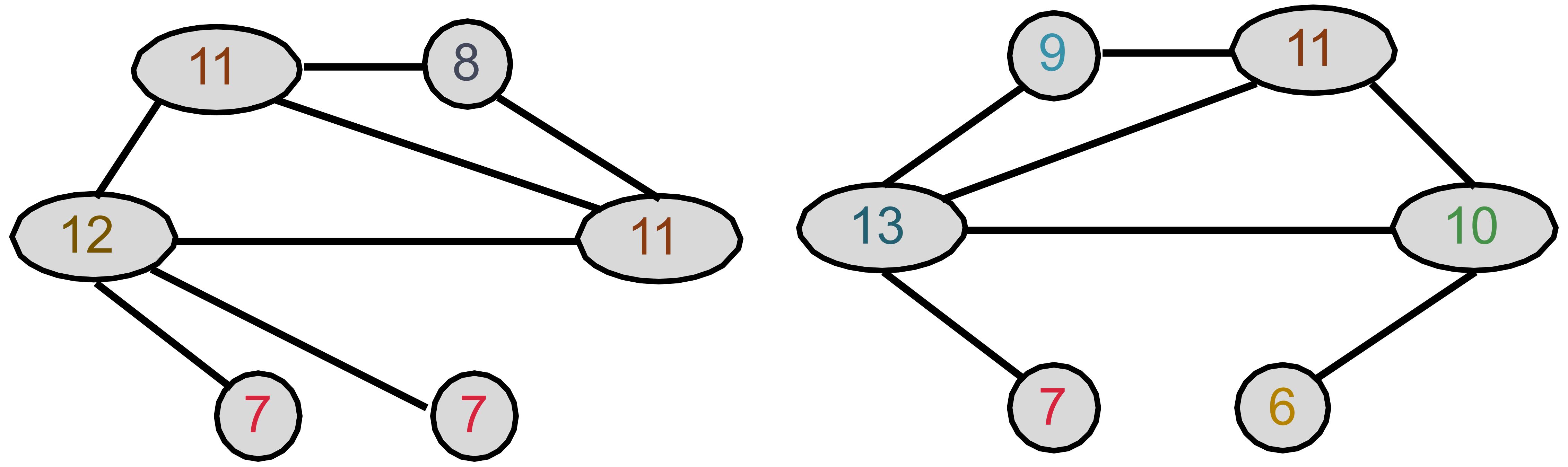
Color Refinement (4)

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

Color Refinement (5)

Stopping Condition:

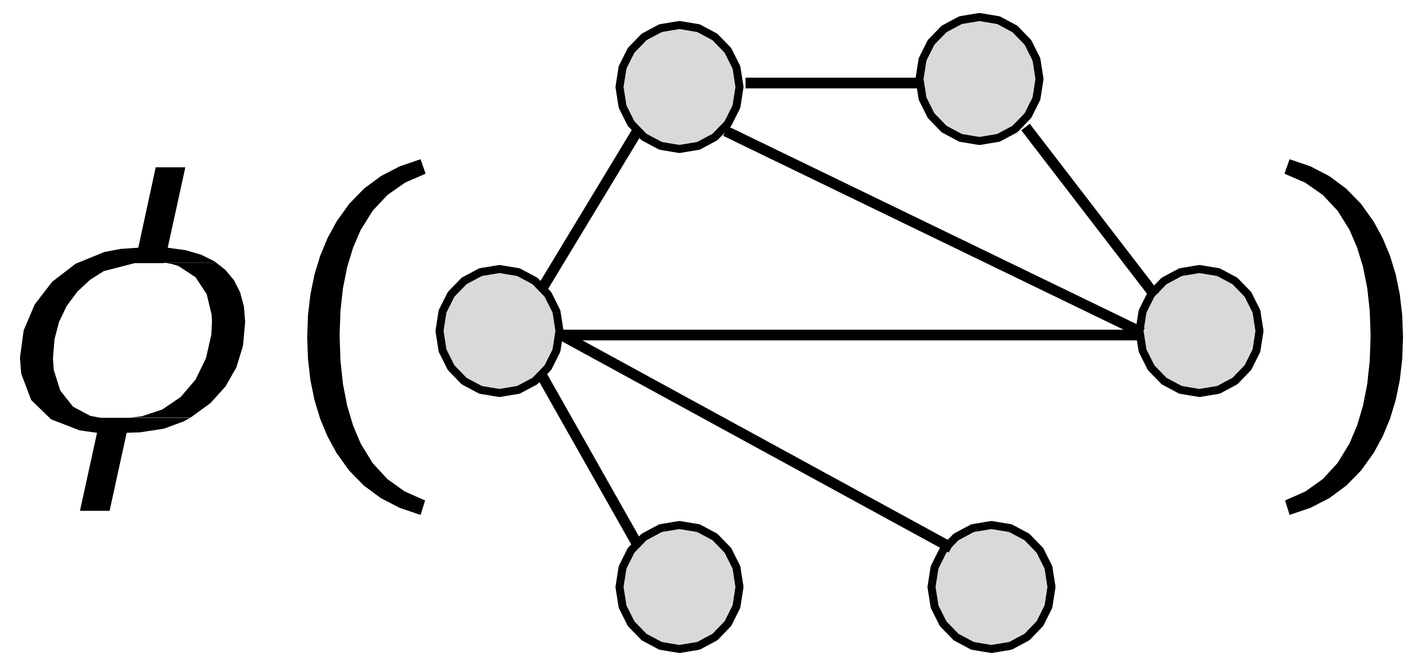
- It stops when **node colors stabilize**.
- Stabilization: **no node's color changes in an iteration based on its current color and neighbors' colors.**

In Practice (for Graph Kernels):

- Often run for a predefined number of iterations (h) for consistent feature vectors.
- Guaranteed to stabilize within the number of nodes.
- Ensure that the resulting feature vectors have the same number of features.
  - NB: the list of unique colors is **global** to all the graphs in the dataset

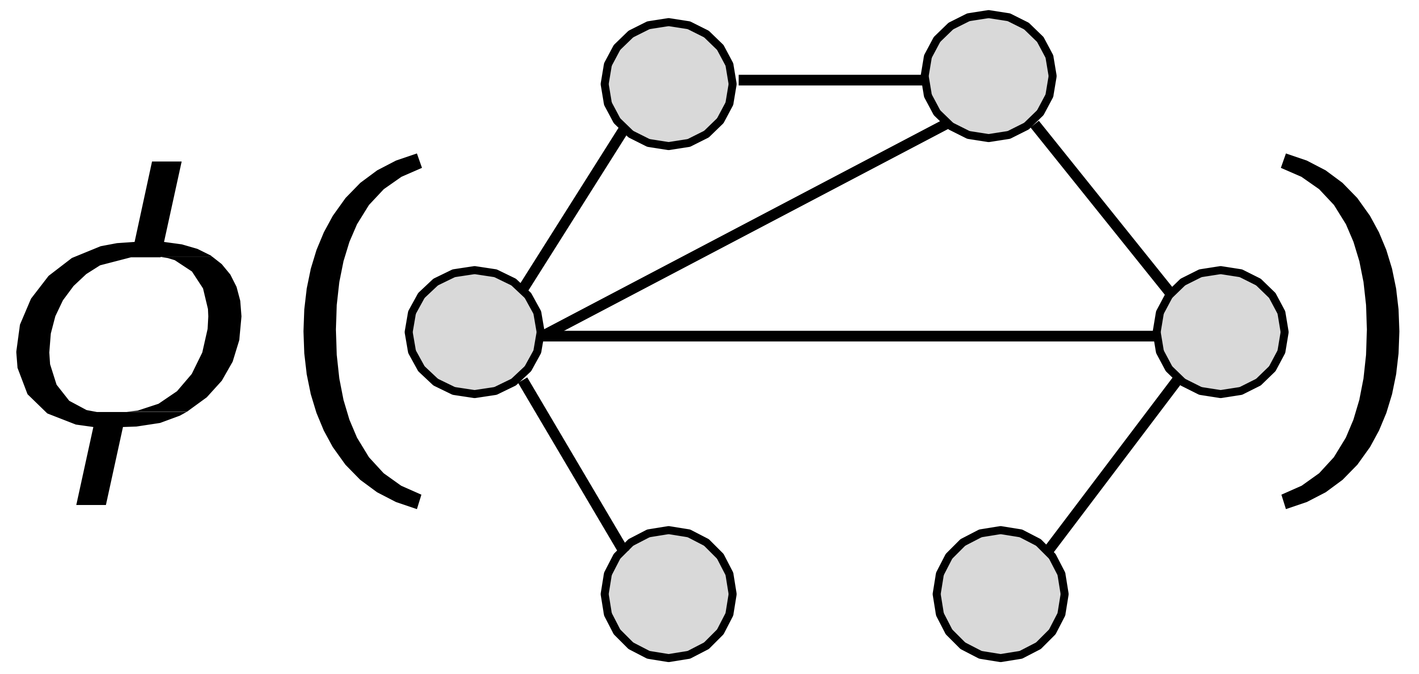
Weisfeiler-Lehman Graph Features

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



1,2,3,4,5,6,7,8,9,10,11,12,13 ← **Colors**

= [6,2,1,2,1,0,2,1,0, 0,2,1,0] ← **Counts**

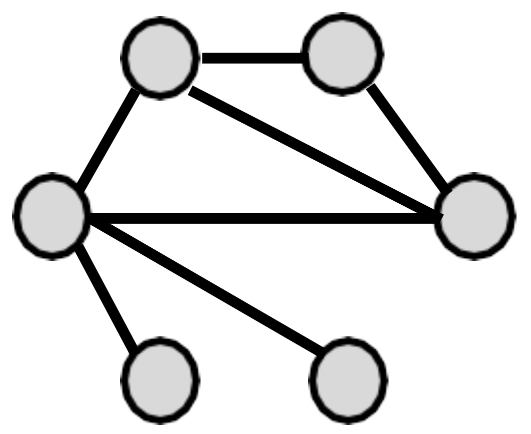
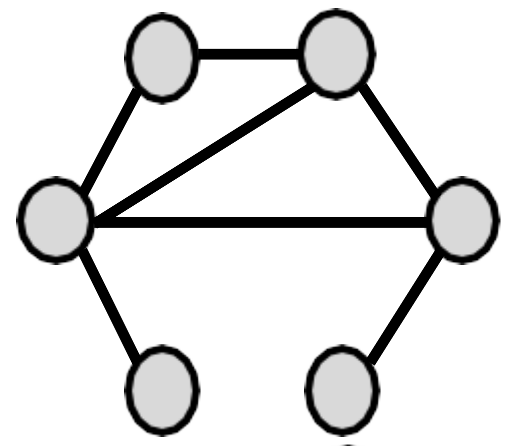


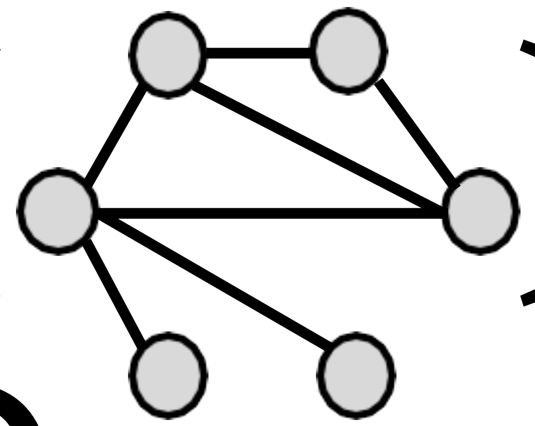
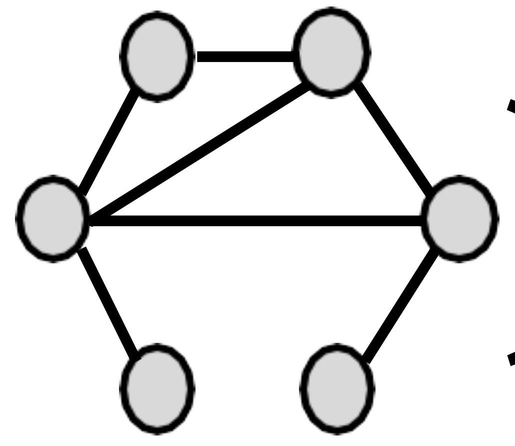
1,2,3,4,5,6,7,8,9,10,11,12,13

= [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:

$K($  $,$  $)$

$= \phi($  $)^{T} \phi($  $)$

$= 49$

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)

UNIVERSITÀ DI TORINO

Q&A