Graph Machine Learning with Python

Dublin, 2025-11-15

> A practical introduction to working with graphs in Python
> https://bit.ly/4dp0Yju

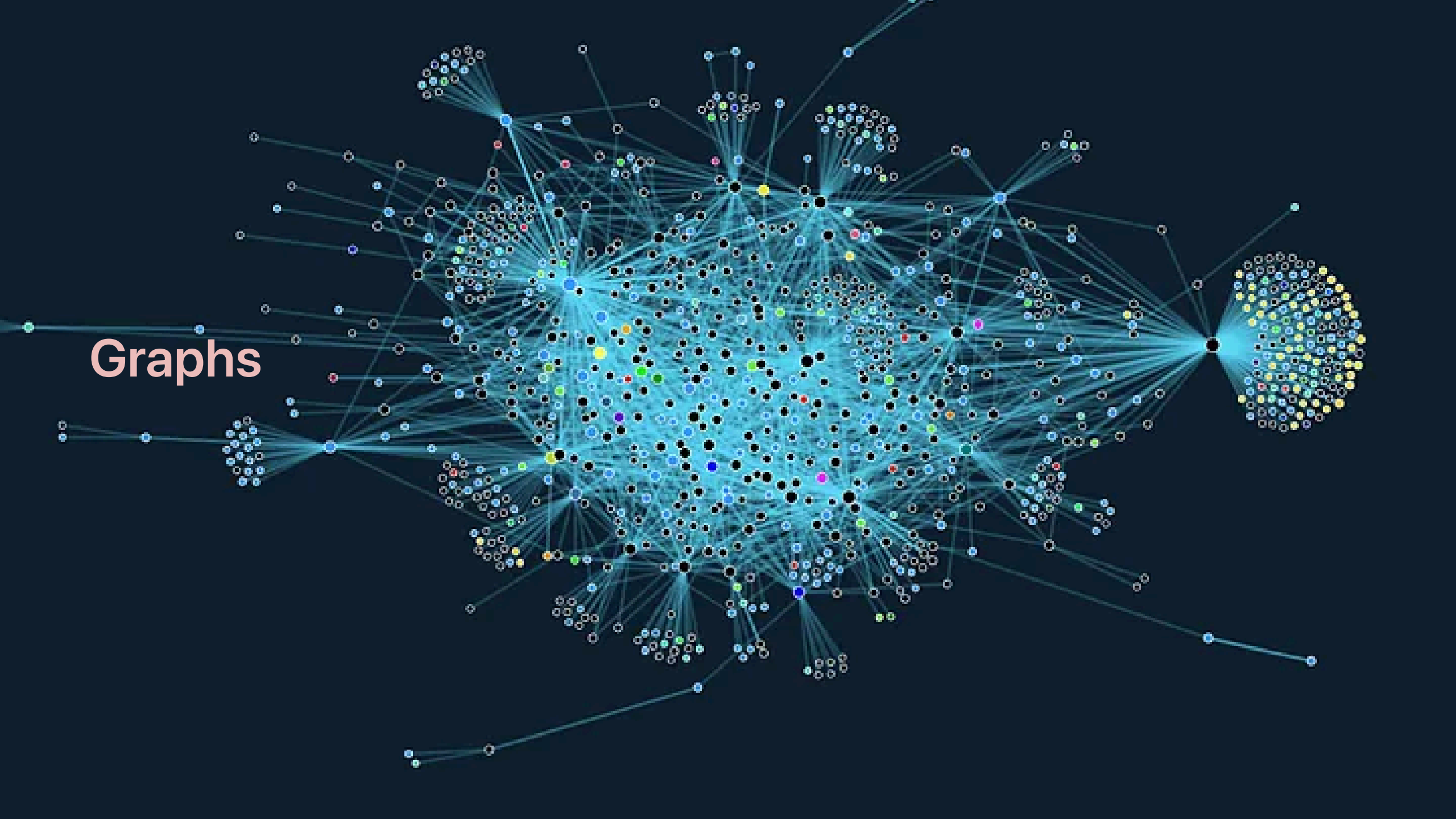
🗂️ Agenda

1. 👋 Introduction
2. 🌐 Graph Fundamentals
3. 🐍 Working with Graphs in Python
4. 🧠 Graph Neural Networks
5. 🛠️ Real world example
6. 🧾 Summary & Q&A

👋 About Me

- 👤 Name: Pietro
- 👨‍👩‍👧 Husband and dad.
- 🇮🇪 Based in Ireland.
- 💼 Data Scientist and AI Engineer.
- 💻 Passionate about applied AI, Python, and Go.
- 🥋 📷 🎛️ 🥾
- 📫 `@iz4vve` (X, github, ...)

Graphs



🌐 What is a Graph?

A **graph** represents the relations (edges) between a collection of entities (nodes).

- **Nodes (or vertices):** entities
- **Edges:** relationships, can have a specific direction.

Both can contain extra information and attributes.

📈 Graph Example

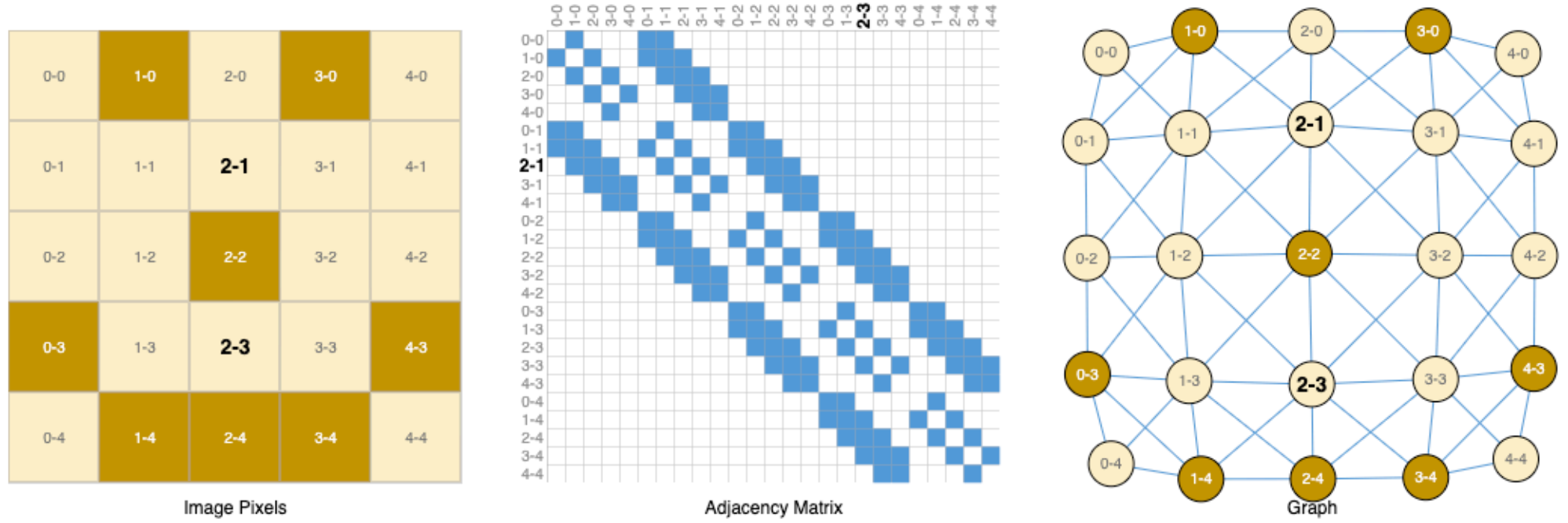
A simple undirected graph showing a small group of individuals and their connections.

🤝 Real-World Graph Examples

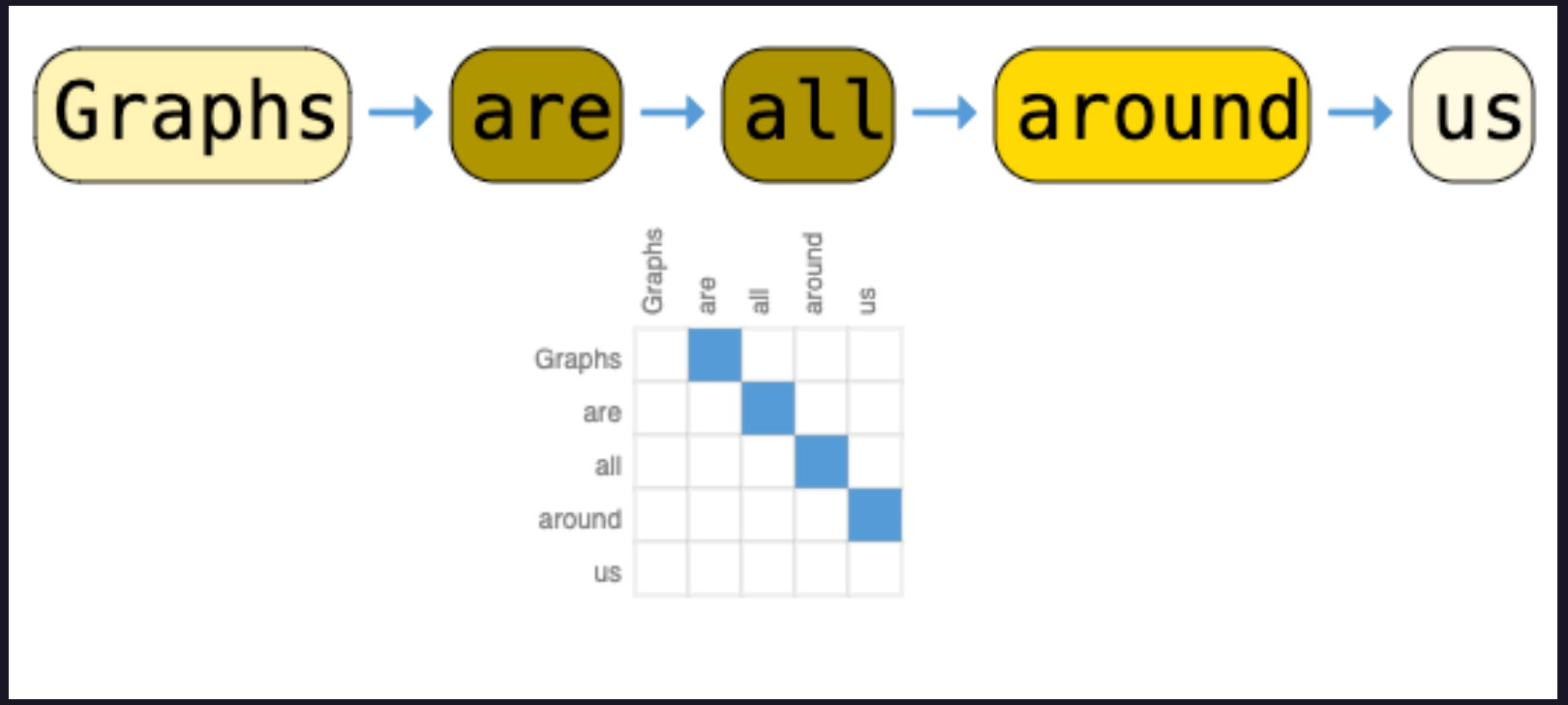
- 👫 Social Networks: People = nodes, Friendships = edges
- 🛒 E-commerce: Products = nodes, "bought-together" = edges
- 💳 Fraud Detection: Accounts = nodes, Transactions = edges
- 🔬 Biology: Proteins = nodes, Interactions = edges

Graphs model relational data — something traditional ML doesn't capture well.

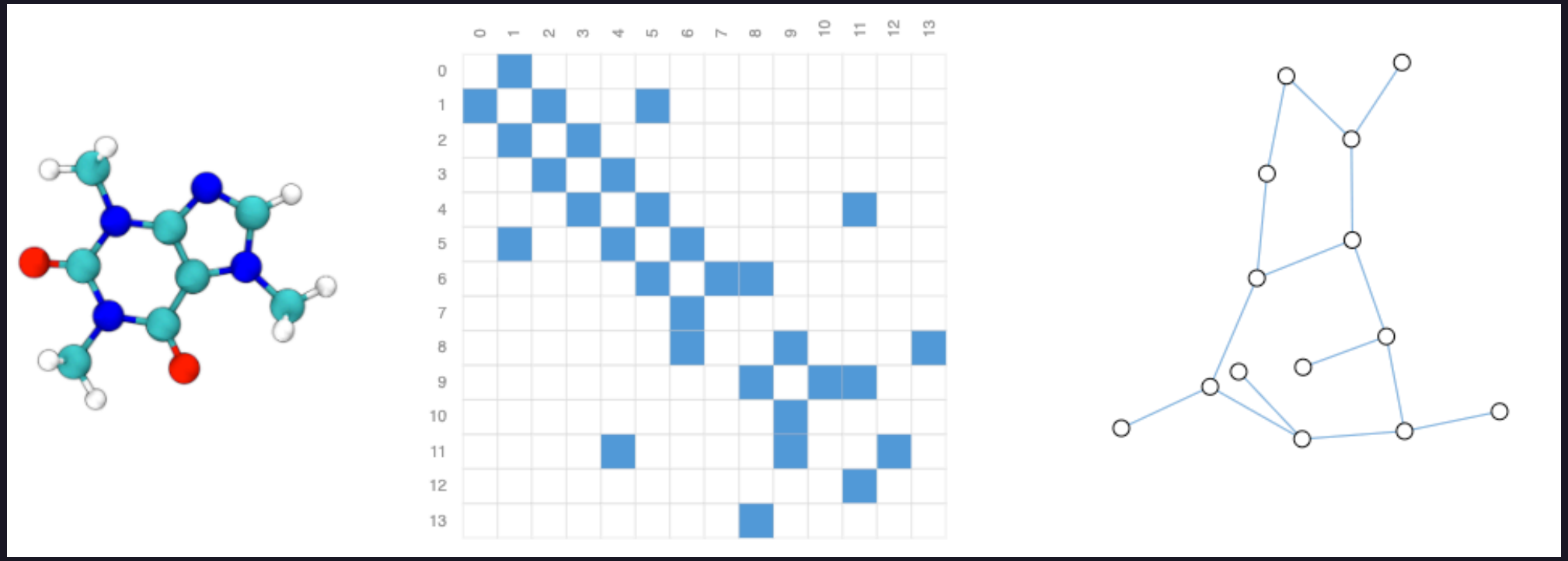
🌍 Graphs are everywhere



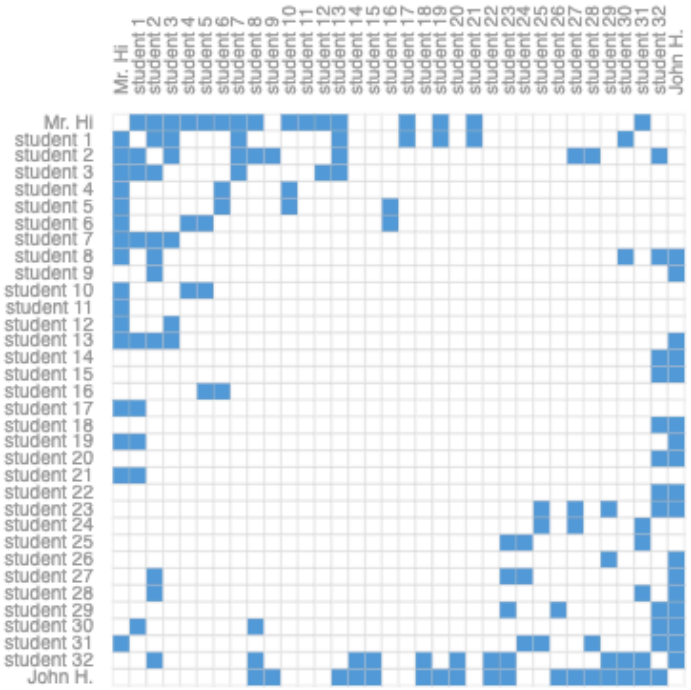
🌍 Graphs are everywhere

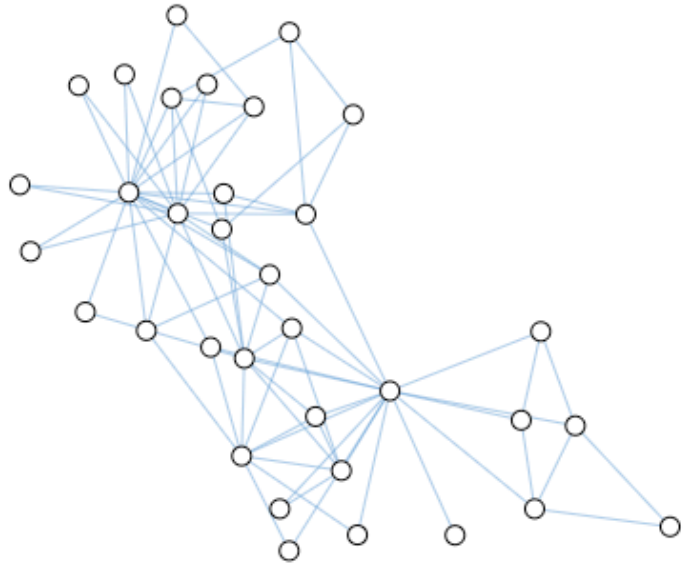


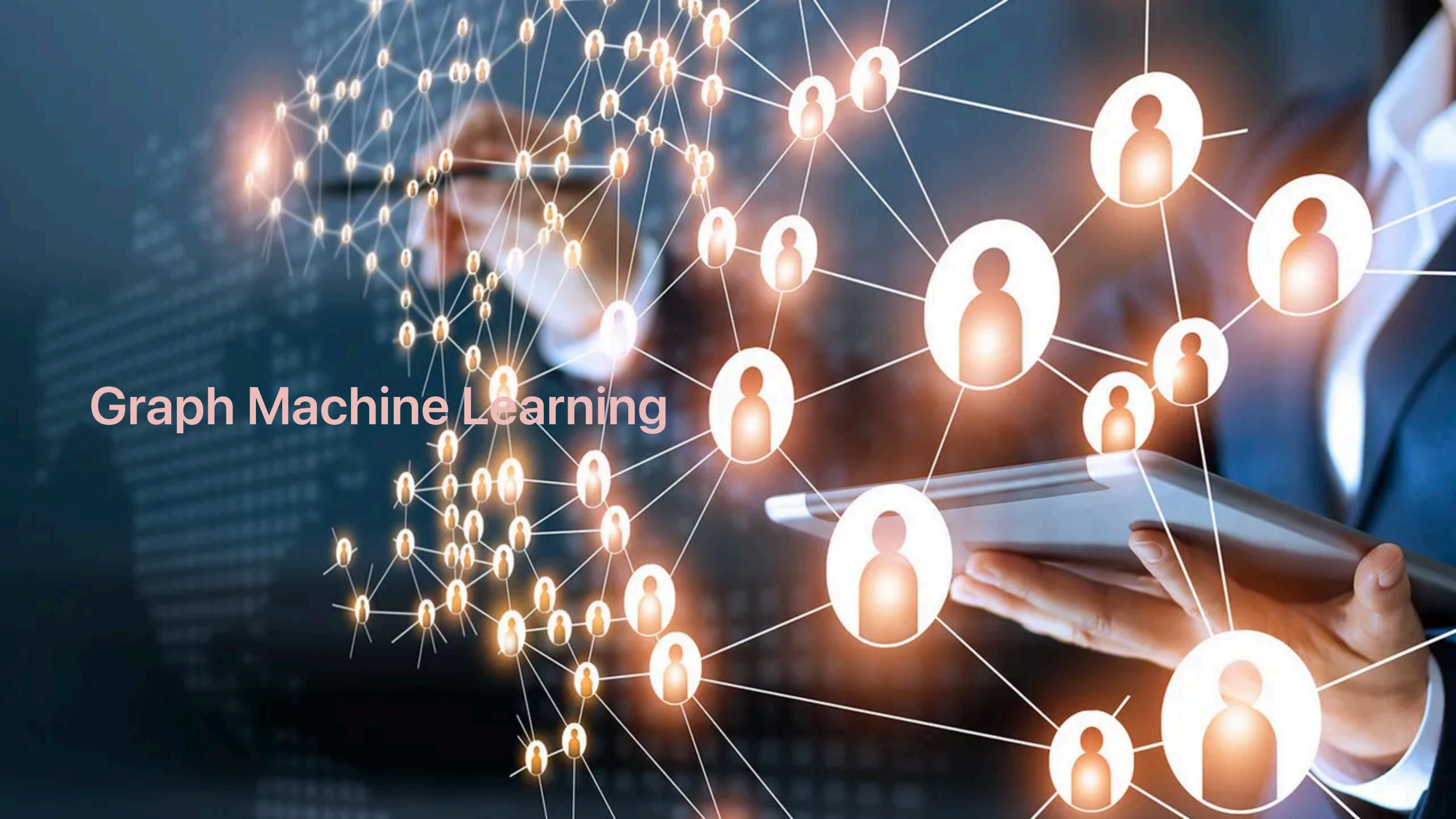
🌍 Graphs are everywhere



🌍 Graphs are everywhere






Graph Machine Learning

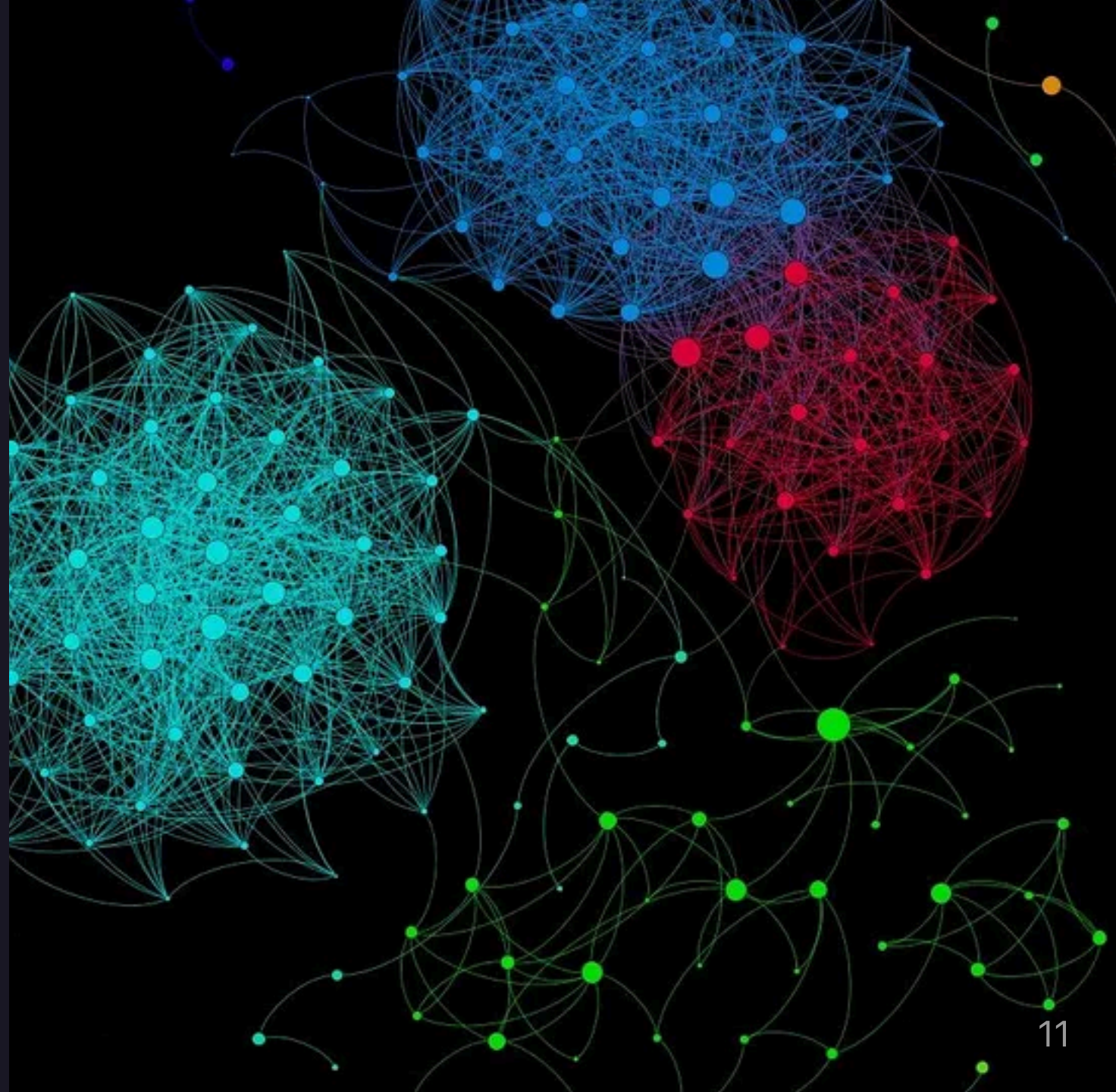
🤔 Why Use Graphs in Machine Learning?

Traditional ML struggles with non-Euclidean data.
Graphs let us:

- Understand **structure and relationships**
- Use **message passing** to learn from context
- Solve tasks like:
  - Node classification (e.g., fraud or not)
  - Link prediction (e.g., will these users connect?)
  - Graph classification (e.g., toxic molecule or not)
  - Analysis of complex relationships

🚩 Types of problems

- Node classification
- Graph classification
- Link prediction
- Graph regression
- Node regression
- Edge regression
- Edge classification
- Spatio-temporal Prediction
- Graph Generation
- ...

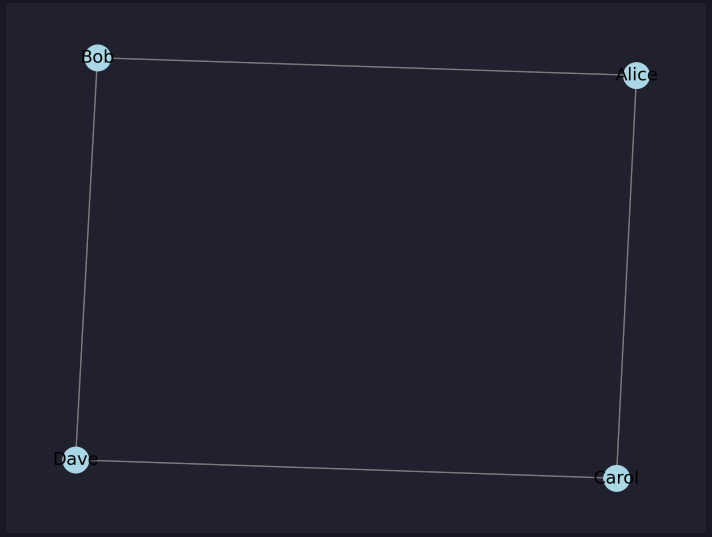


Graphs with `networkx`

```python
import networkx as nx

G = nx.Graph()
G.add_edges_from([
    ("Alice", "Bob"), ("Alice", "Carol"), ("Carol", "Dave"), ("Bob", "Dave")
])

nx.draw(G, with_labels=True, node_color="lightblue", edge_color="gray")
```



🔍 Graph Analysis with `networkx`

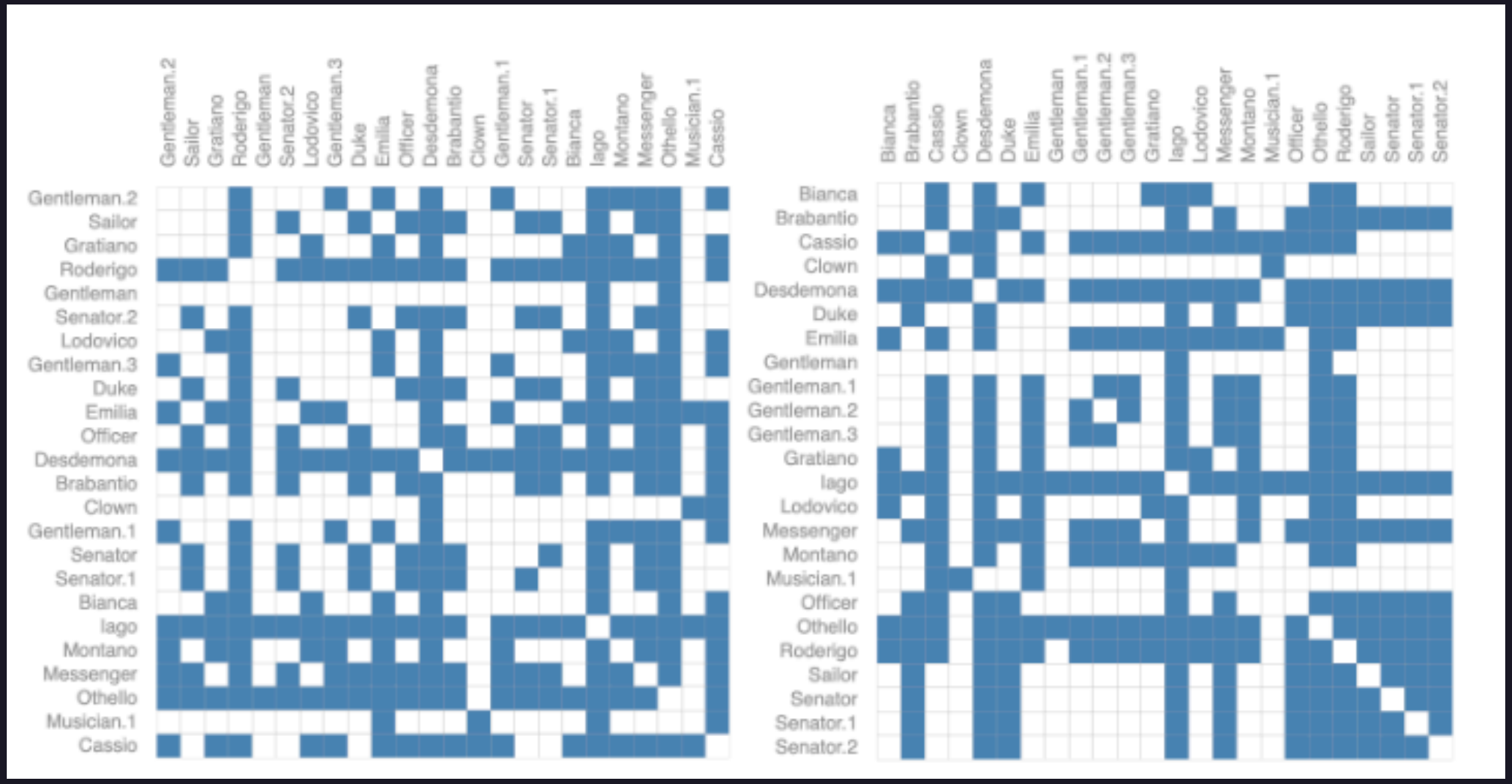
```
>>> print("Nodes:", G.nodes())
Nodes: ['Alice', 'Bob', 'Carol', 'Dave']

>>> print("Edges:", G.edges())
Edges: [('Alice', 'Bob'), ('Alice', 'Carol'), ('Bob', 'Dave'), ('Carol', 'Dave')]

>>> print("Degree of Carol:", G.degree("Carol"))
Degree of Carol: 2

# Find shortest path
>>> path = nx.shortest_path(G, source="Alice", target="Dave")
>>> print("Shortest path from Alice to Dave:", path)
Shortest path from Alice to Dave: ['Alice', 'Carol', 'Dave']
```

🎨 Graph representation



🎨 Graph representation (permutation invariant)

```
# adjacency list
adj_list = """
  A B C
  B C D
  C D
  D A
"""

node_properties = {
  "A": { ... },
  "B": { ... },
  ...
}

edge_properties = {
  ('A', 'B'): { ... },
  ('A', 'C'): { ... },
  ...
}
```

🧠 What Are Graph Neural Networks (GNNs)?

> **GNNs are neural networks that operate on graph structures.**

Each node aggregates and updates its feature vector from its neighbors.

Typical pipeline:

1. Initialize node features (e.g., account age, balance)
2. Perform message passing via GNN layers
3. Predict labels or scores (fraud, risk, etc.)

📚 Popular GNN Architectures

- **GCN** (Graph Convolutional Network)
- **GAT** (Graph Attention Network)
- **GraphSAGE**
- **GIN** (Graph Isomorphism Network)
- ...

Libraries like PyTorch Geometric make these easy to use.

🧠 How Do Graph Neural Networks Work?

A GNN learns a **representation for each node** based on:

- Its own features
- Its neighbors' features

This is done through a process called Message Passing

🔄 Message Passing – Step-by-Step

2️⃣ At Each Layer:

- A node gathers messages from neighbors
- Aggregates them (e.g. sum, mean)
- Updates its own feature using a neural net

🧠 GCN Layer

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} W h_u^{(l)}\right)$$

Where:

- $(h_v^{(l)})$: Node (v)'s features at layer (l)
- $(\mathcal{N}(v))$: Neighbors of (v)
- (W): Learnable weight matrix
- (σ): Non-linearity (e.g. ReLU)

🔁 Recap: GNNs in a Nutshell

✅ **GNNs learn node embeddings**
✅ **Each layer mixes neighborhood information**
✅ **Useful for:**

- Node classification (e.g. fraud detection)
- Link prediction (e.g. friend suggestion)
- Graph classification (e.g. molecule toxicity)

Implementing a GNN

🧪 Simple GNN with PyG

```python
from torch_geometric.nn import GCNConv
import torch.nn.functional as F
import torch


class GCN(torch.nn.Module):
    def __init__(self, num_features=100, num_classes=2, hidden_dim=16):
        super().__init__()
        self.conv1 = GCNConv(num_features, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, num_classes)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

🏋️ Train the GNN

```
model = GCN()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.NLLLoss()

model.train()
for epoch in range(200):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = criterion(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()

print("Training complete.")
```

🔬 Evaluate the GNN

```
model.eval()
pred = out.argmax(dim=1)
correct = (pred[data.test_mask] == data.y[data.test_mask]).sum()
accuracy = int(correct) / int(data.test_mask.sum())
print(f"Test Accuracy: {accuracy:.4f}")
```

⚠️ Real-World Use Case: Paper classification

Paper citations often form complex **graph topologies**.

A GNN can learn **structural patterns** better than traditional ML, which can be leveraged to infer a paper's category from the paper it cites and the papers that cite it.

📚 Cora Dataset Overview

A benchmark citation network used for node classification and graph learning tasks.

Property	Value
Dataset Name	Cora
Task	Node Classification
Number of Nodes	2,708
Number of Edges	5,429 (undirected)
Number of Classes	70
Node Features	1,433 (bag-of-words)
Edge Type	Paper cites paper

💡 Label Distribution & Use

- Each node = 1 paper
- Classes = 7 topics (e.g., Reinforcement Learning, Probabilistic Methods)
- Used in benchmarks for: GCN, GAT, GraphSAGE, etc.

Results

	Training time	Accuracy	Precision	Recall	F1	AUROC (OvR)
GBC	218 min	0.543	0.554	0.543	0.542	0.904
GCN	27.8 s	0.722	0.722	0.723	0.722	0.985

🧾 Conclusions

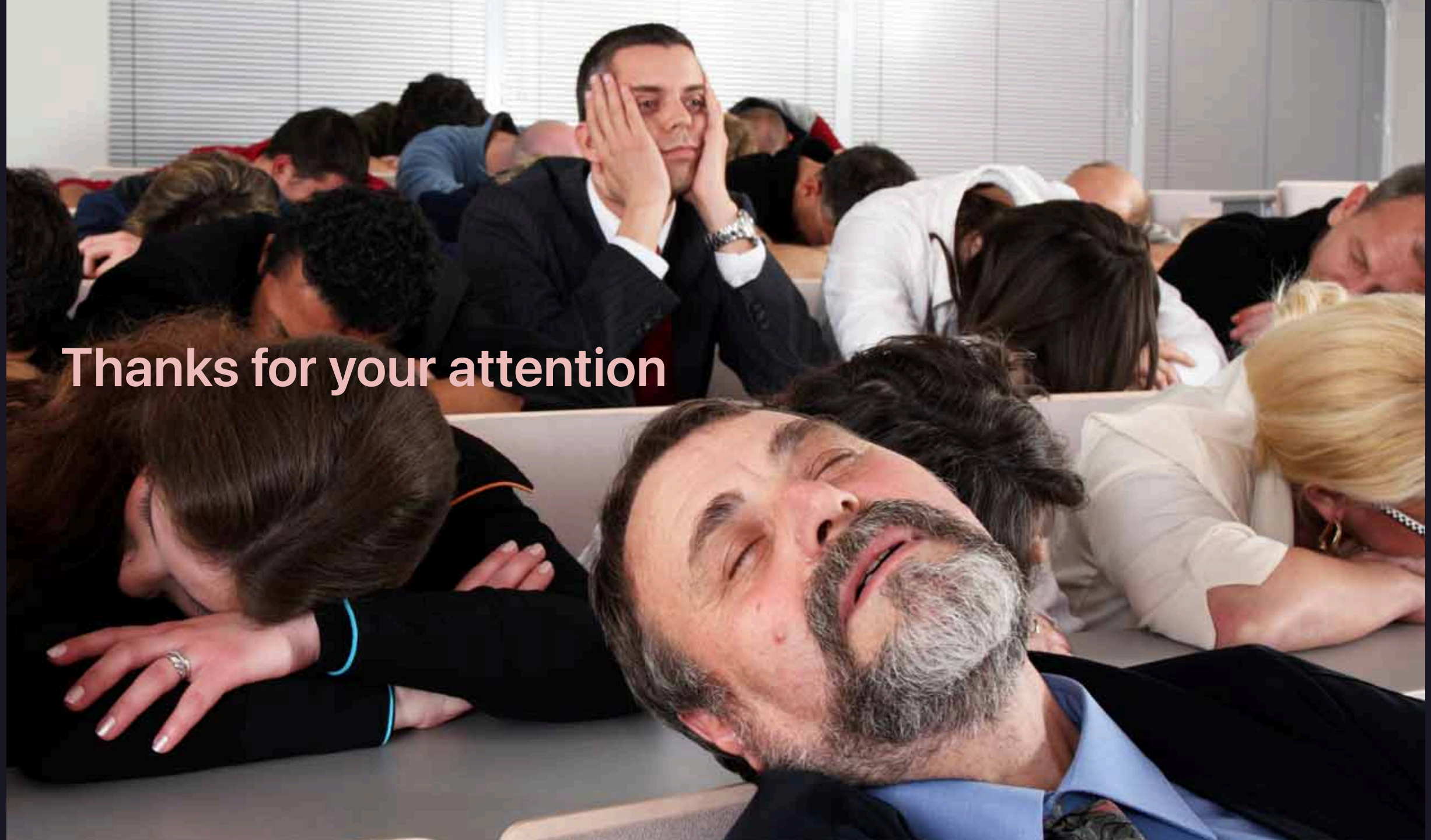
- GNNs **capture structure** — they understand not just what something is, but **how it's connected**
- Real-world data is often **relational**

GNNs are useful when connections matter more than just features:

- **Fraud detection** – suspicious behavior is often found in clusters
- **Recommender systems** – based on user-item relationships
- **Social networks** – influence spreads through links
- **Molecular biology** – atoms connected in meaningful ways

🙋‍♀️🙋🙋‍♂️ Questions?

You can ask... I might even answer...


Thanks for your attention