Overview of RAPIDS Graph and advice on graph creation

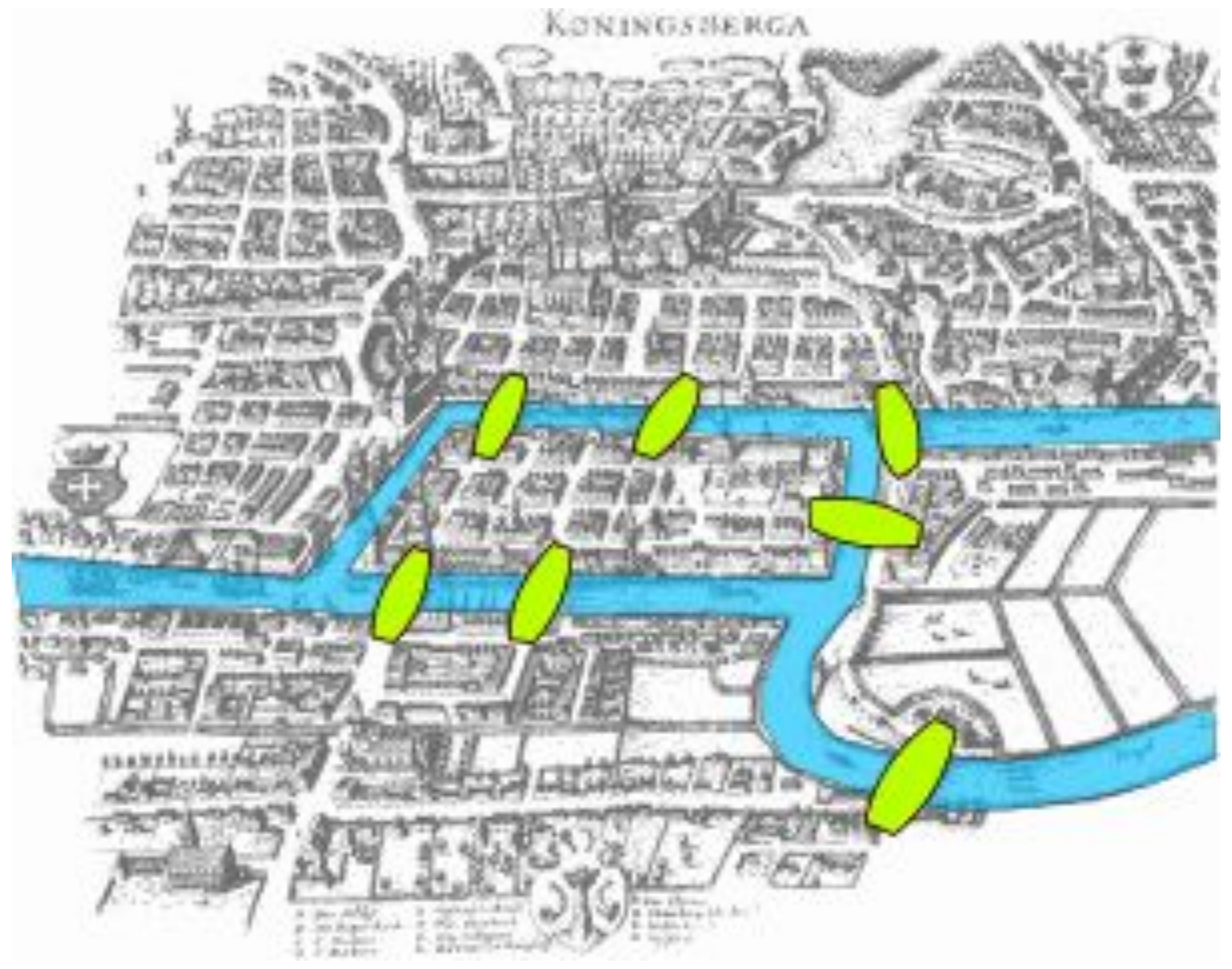
Topics covered

- Overview of cuGraph
- Build a bad graph, get a bad answers

Graph is Not a New Concept

Its always good to look back on the history of graph

- The “Seven Bridges of Königsberg” – the problem that started it all
  - Euler’s solution laid the foundation for Graph Theory in 1736
    - Traveling Salesman Problem, graph traversal
- “Six Degrees of Kevin Bacon” game - 2000-ish
  - “Collective dynamics of ‘small-world’ networks by Duncan Watts & Steven Strogatz – 1998
    - “The small-world problem” by Stanley Milgram - 1968
      - “Chains” by Karinthy Frigyyes - 1928
        - “Around the World in Eighty Days” – Jules Verne – 1872
- Social Network
  - Not it was not invented by Facebook
    - The term “social network” was first used by sociologist John Arundel Barnes in 1954
      - Prior to that was “Sociograph” used by J.L. Moreno in 1933
        - Moreno hand drew graphs with millions of nodes
- Many of our problems have also been around for year
  - “Fake news and the public” – Harper’s Magazine – 1925
  - Money Laundering dates by 2000+ years
- We are developing new techniques
  - However, bad actors are also developing new techniques



Ref: https://en.wikipedia.org/wiki/Seven_Bridges_of_Königsberg

Why *RAPIDS Graph* and not *cuGraph*

- We launched RAPIDS in 2018 with cuGraph being the library for graph analytics and project name
- But the project has grown since then

cuGraph Core
cugraph
pylibcugraph
libcugraph_c
libcugraph
libcugraph_etl

GNN Packages
cugraph-dgl
cugraph-pyg
wholegraph

Other Packages
nx-cugraph
cugraph-service

Other Work
Research
Customer engagements
Documentation

Equivariance worked moved into BioNemo

Why *RAPIDS Graph* and not *cuGraph*

- We launched RAPIDS in 2018 with cuGraph being the library for graph analytics and project name
- But the project has grown since then

Joe's talk

cuGraph Core
cugraph
pylibcugraph
libcugraph_c
libcugraph
libcugraph_etl

GNN Packages
cugraph-dgl
cugraph-pyg
WholeGraph

Other Packages
nx-cugraph
cugraph-service

Other Work
Research
Customer engagements
Documentation

Equivariance worked moved into BioNemo

Lots of Integration Options

Six Integration Points

1) cuGraph   Python: rich layer, dependent on cuDF and DASK
2) pylibcugraph   Python: light weight, minimal guard rails, 2-10x faster
3) libcugraph_c   C API
4) libcugraph   C++ API
    1) MTMG C++ API for multi-threaded application integration
5) cugraph-service   Graph-as-a-service

cuGraph Core
cugraph
pylibcugraph
libcugraph_c
libcugraph
libcugraph_etl

Benefits of Integrating cuGraph

- Fast GPU Accelerated Graph Algorithms
    - Setting a new standard of performance
- Scalable to address any size problem
- Constant Improvement
- New algorithms being added

Common graph primitive layer: GrAPL 2023
"*cuGraph C++ primitives: vertex/edge-centric building blocks for parallel graph computing*,"

Evolution of Accelerated Computing

Finding the right niche for every kind of user

nx-cugraph →

Easie
Use

Zero Code Change: Acceleration Plugins *(no-code change)*

cudf.pandas: Accelerated Pandas, nx-cugraph: Accelerated NetworkX, RAPIDS Spark Accelerator, Array-API backed Scikit-learn, …

Hybrid CPU/GPU libraries *(minimal change)*

Pytorch, FAISS, Tensorflow, XGBoost, cuML-CPU, Dask, pySpark, …

cugraph →

GPU Python Libraries *(GPU Python code)*

RAPIDS core libraries (RMM, cuDF, cuML, cuGraph, cuVS…), CuPy, Numba, OpenAI Triton, …

pylibcugraph →

Python/CUDA libraries *(Hybrid Python / CUDA code)*

CuPy RawKernels, Numba CUDA, Cython wrappers for CUDA, …

libcugraph →

C++/CUDA high level *(High-level C++/CUDA code)*

RAFT, CCCL (Thrust, CUB, libcucxx), cuBLAS, cuDNN, cuSolver, cuSPARSE, …

CUDA Toolkit *(C++/CUDA code and kernels)*

Raw CUDA kernels

Higher
Performance

Current List of Algorithms

Class	Algorithms	MNMG
Centrality	Katz	Yes
	Betweenness Centrality	Yes
	Edge Betweenness Centrality	Yes
	Eigenvector Centrality	Yes
	Degree Centrality (Python only)	Yes
Community	Leiden	Yes
	Louvain	Yes
	Ensemble Clustering for Graphs	
	Spectral-Clustering - Balanced Cut	
	Spectral-Clustering - Modularity	
	Subgraph Extraction	Yes
	Triangle Counting	Yes
	K-Truss	Yes
Components	Weakly Connected Components	Yes
	Strongly Connected Components	
Core	K-Core	Yes
	Core Number	Yes

Class	Algorithms	MNMG
Layout	Force Atlas 2	not planned
Link Analysis	PageRank	Yes
	Personal PageRank	Yes
	HITS	Yes
Link Prediction / Similarity	Jaccard Similarity	Yes
	Weighted Jaccard Similarity	Yes
	Overlap Similarity	Yes
	Sorensen	Yes
Traversal	Breadth First Search (BFS)	Yes
	Single Source Shortest Path (SSSP)	Yes
Sampling	Random Walks (Uniform and Biased)	Yes
	EgoNet	Yes
	Node2Vec	Yes
	Neighborhood sampling	Yes
Other	Minimum/Maximum Spanning Tree	not planned
	Hungarian	not planned
	RMAT	Yes

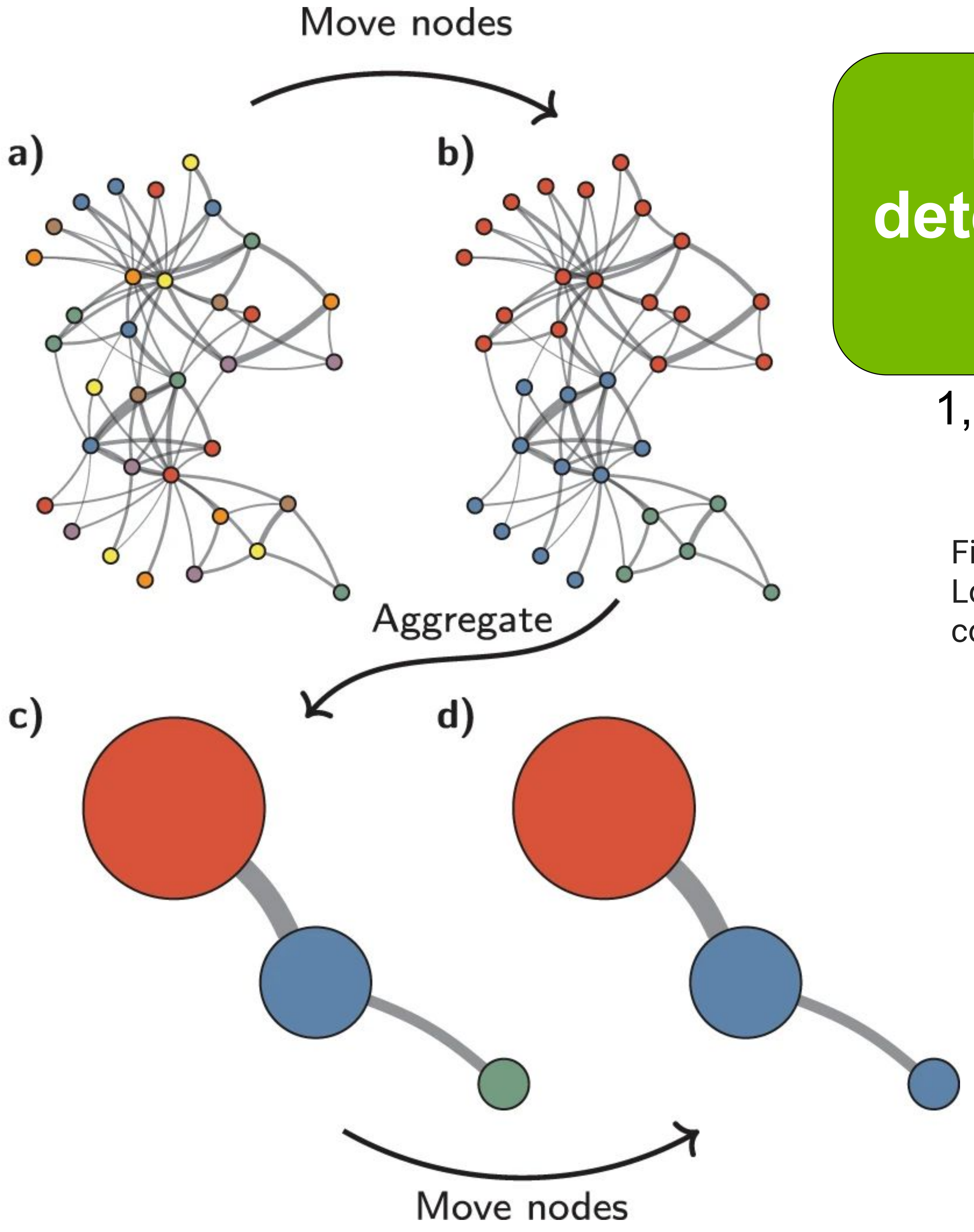
Always eager to hear what customer want.
We add new algorithms based on customer request

Current focus has been on Sampling algorithms for GNNs
Coming soon: Heterogeneous and Temporal Sampling

CUGRAPH SCALING

TACKLING THE WORLD'S LARGEST GRAPHS (Reflecting on Old Work)

- New 2-D data partitioning scheme for scalability
- Modular Graph Primitives
  - hide complexity of data partitioning
  - allows all algorithms to be built on a common set of functions
- Testing and benchmarking at supercomputer scales
  - Constant testing all MNMG algorithms up to 64 GPUs



Louvain community detection on 64B edges in ~90s

1,536 GPUs at Oak Ridge (2021)

Figure: Traag, V.A., Waltman, L. & van Eck, N.J. From Louvain to Leiden: guaranteeing well-connected communities. *Sci Rep* 9, 5233 (2019).

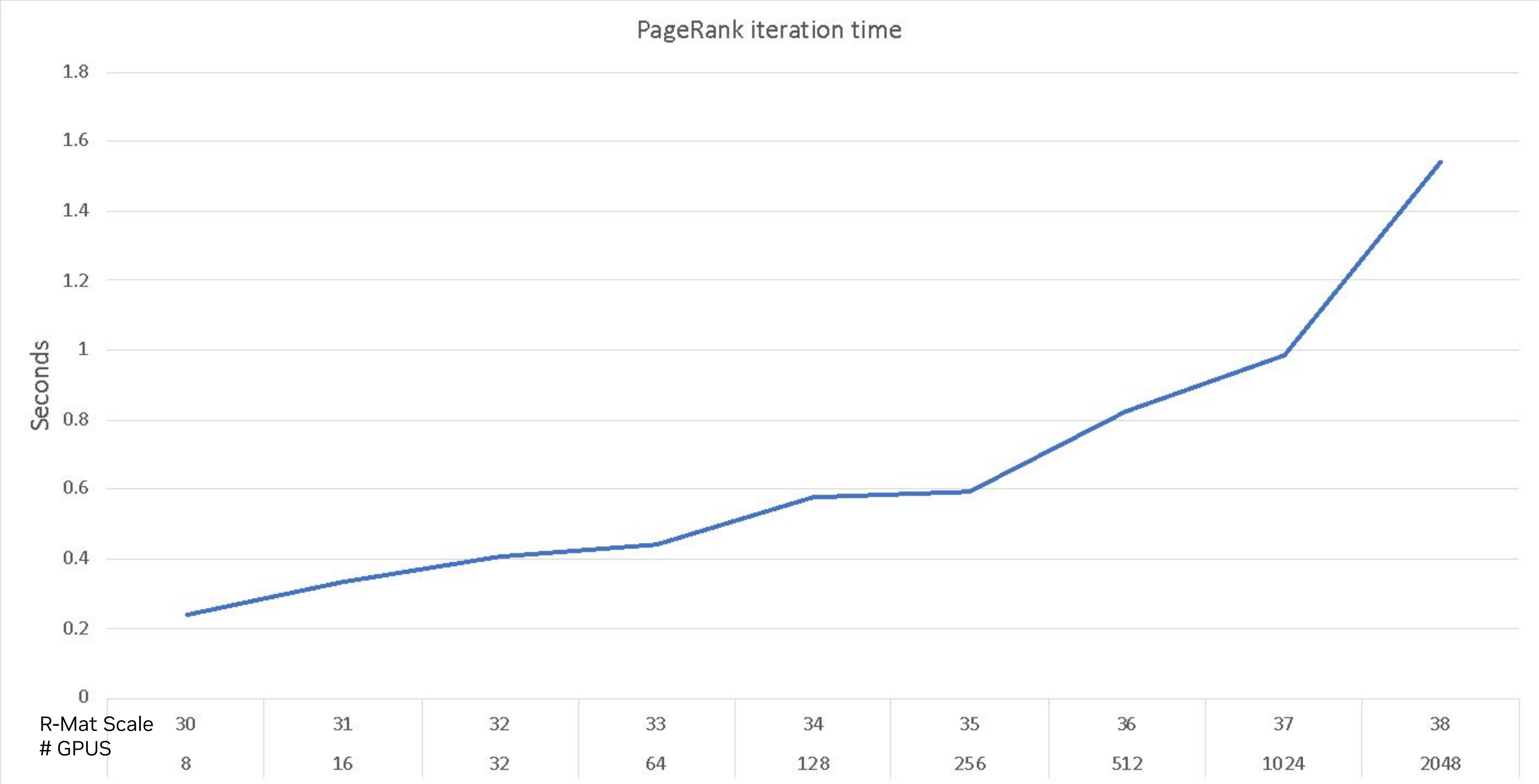
PageRank on 128B edges at 0.187s/iteration

32 GPUs (4 DGXA100) (2020)

S. Kang, A. Fender, J. Eaton and B. Rees, "Computing PageRank Scores of Web Crawl Data Using DGX A100 Clusters," *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1-4, doi: 10.1109/HPEC43674.2020.9286216.

Performance and Scalability

The newer stuff



Scaling (C++)

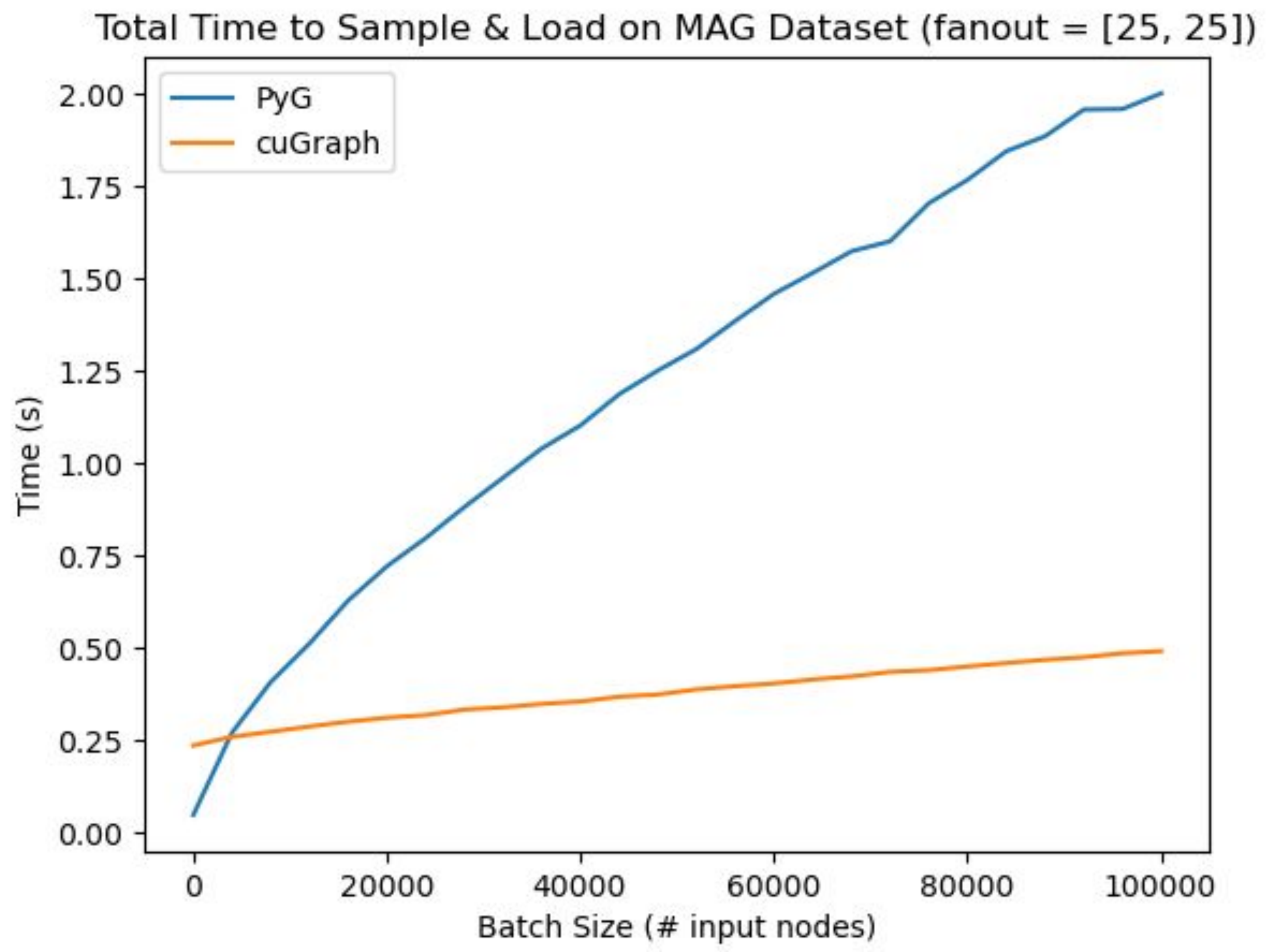
- **PageRank:**
  - **Scale 36** (1.1 trillion directed edges) in 19.3 seconds (0.66 seconds per iteration, 2,048 GPUs)
  - **Scale 38** (**4.4 trillion edges**) at 1.54 seconds per iteration on 2,048 GPUs
- **Louvain:** Scale 35 (0.55 trillion undirected edges or 1.1 trillion directed edges) in 336 seconds (1024 GPUs)

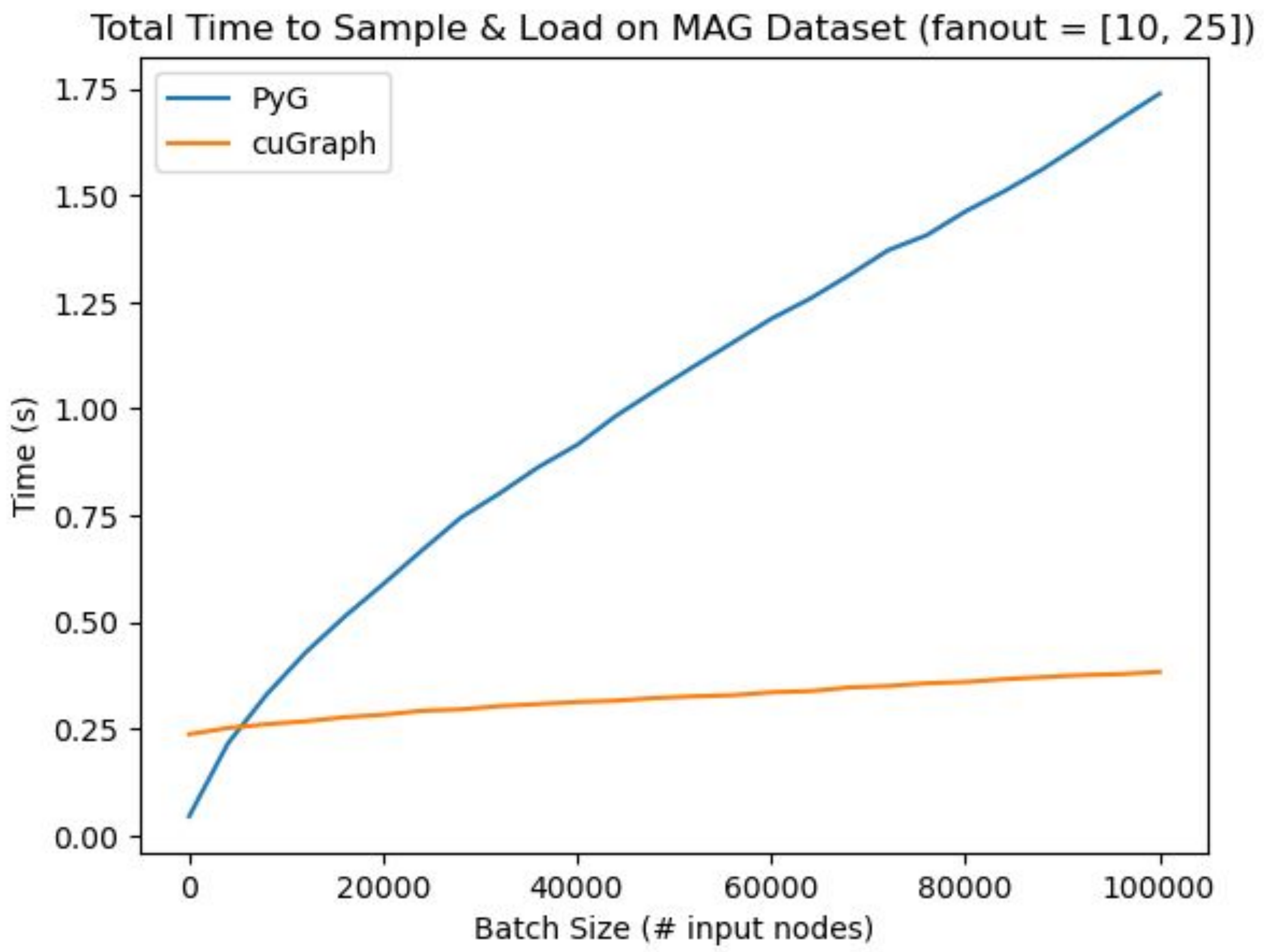
Scale	Number of Vertices	Number of Edges	COO Data Size (GB) in GPU
28	268,435,456	4,294,967,296	80
29	536,870,912	8,589,934,592	160
30	1,073,741,824	17,179,869,184	320
31	2,147,483,648	34,359,738,368	640
32	4,294,967,296	68,719,476,736	1,280
33	8,589,934,592	137,438,953,472	2,560
34	17,179,869,184	274,877,906,944	5,120
35	34,359,738,368	549,755,813,888	10,240
36	68,719,476,736	1,099,511,627,776	20,480
37	137,438,953,472	2,199,023,255,552	40,960

HPEC 2021: *Analyzing Multi-trillion Edge Graphs on Large GPU Clusters: A Case Study with PageRank*

Graph Sampling

- Multiple Sampling Algorithms
  - Egonet
  - Random Walk
  - Node2Vec
  - Neighborhood





- Our algorithm scales close to linear with the number of seeds.
  - We can sample 100K seeds just as fast as sampling 100 seeds

- Joe will discuss how we use pre-fetching of samples for GNNs

The cuGraph API

- cuGraph had a vision of being a drop-in replacement for NetworkX
  - A grand vision that is not easily achieved
- Fundamental d ... gence in API
  - We dropped ... ted
  - A path from ... ide

```
import networkx as nx
import time
import operator

# create a random graph
G = nx.barabasi_albert_graph(N, M)

... do some NetworkX stuff ..

t1 = time.time()
bc = nx.betweenness_centrality(G)
t2 = time.time() - t1

print(t2)
```

NetworkX

```
import networkx as nx
import time
import operator
import cugraph as cnx

# create a random graph
G = nx.barabasi_albert_graph(N, M)

... do some NetworkX stuff ..

t1 = time.time()
bc = cnx.betweenness_centrality(G)
t2 = time.time() - t1

print(t2)
```

NetworkX + RAPIDS cuGraph

Node	Edges	Speedup
100	1,344	0.45
200	2,944	1.14
400	6,144	2.64
800	12,544	5.26
1,600	25,344	12.99
3,200	50,944	26.5
6,400	102,144	48.62
12,800	204,544	89.81
25,600	409,344	180.42
51,200	818,944	328.05

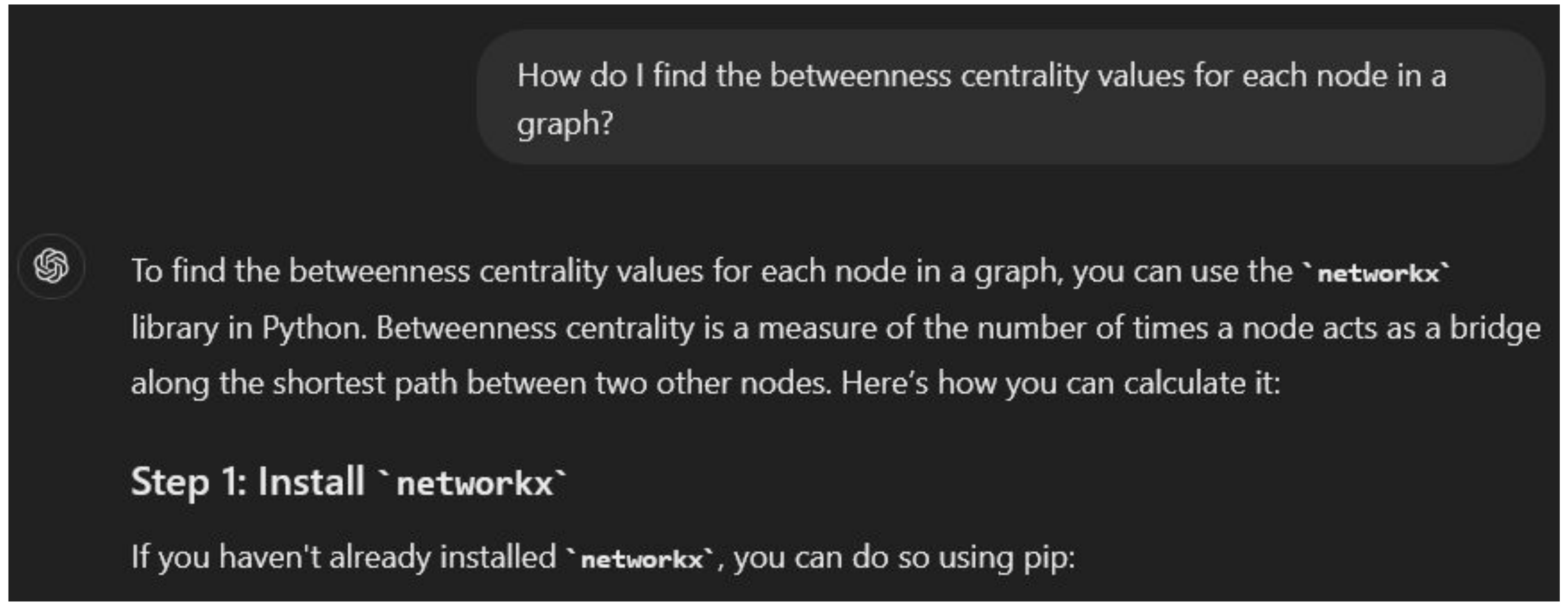
The cuGraph API

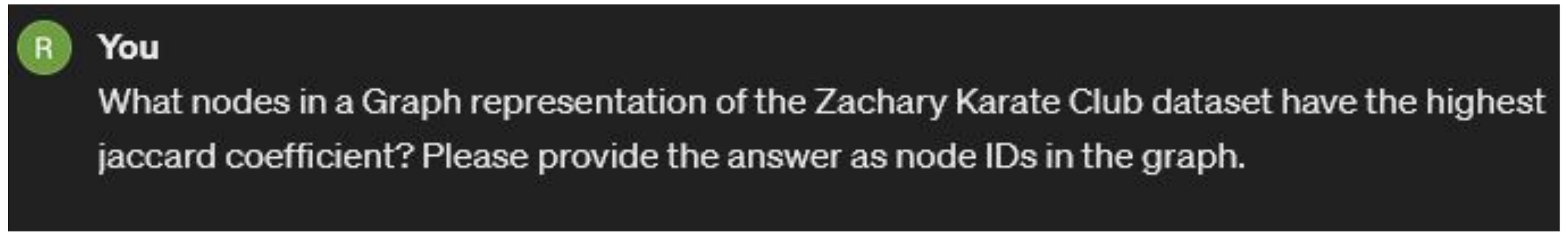
- cuGraph had a vision of being a drop-in replacement for NetworkX
  - A grand vision that is not easily achieved
- Fundamental differences in data storage, scalability, and integration with other RAPIDS effort caused to a divergence in API
  - We dropped being a “drop-in” replacement and shifted to be “NetworkX-like”
  - A path from NetworkX to cuGraph was added as a sidecar with meh performance
- We always wanted to truly work with NetworkX
  - Luckily, we are persistent and take a long-term view of problems

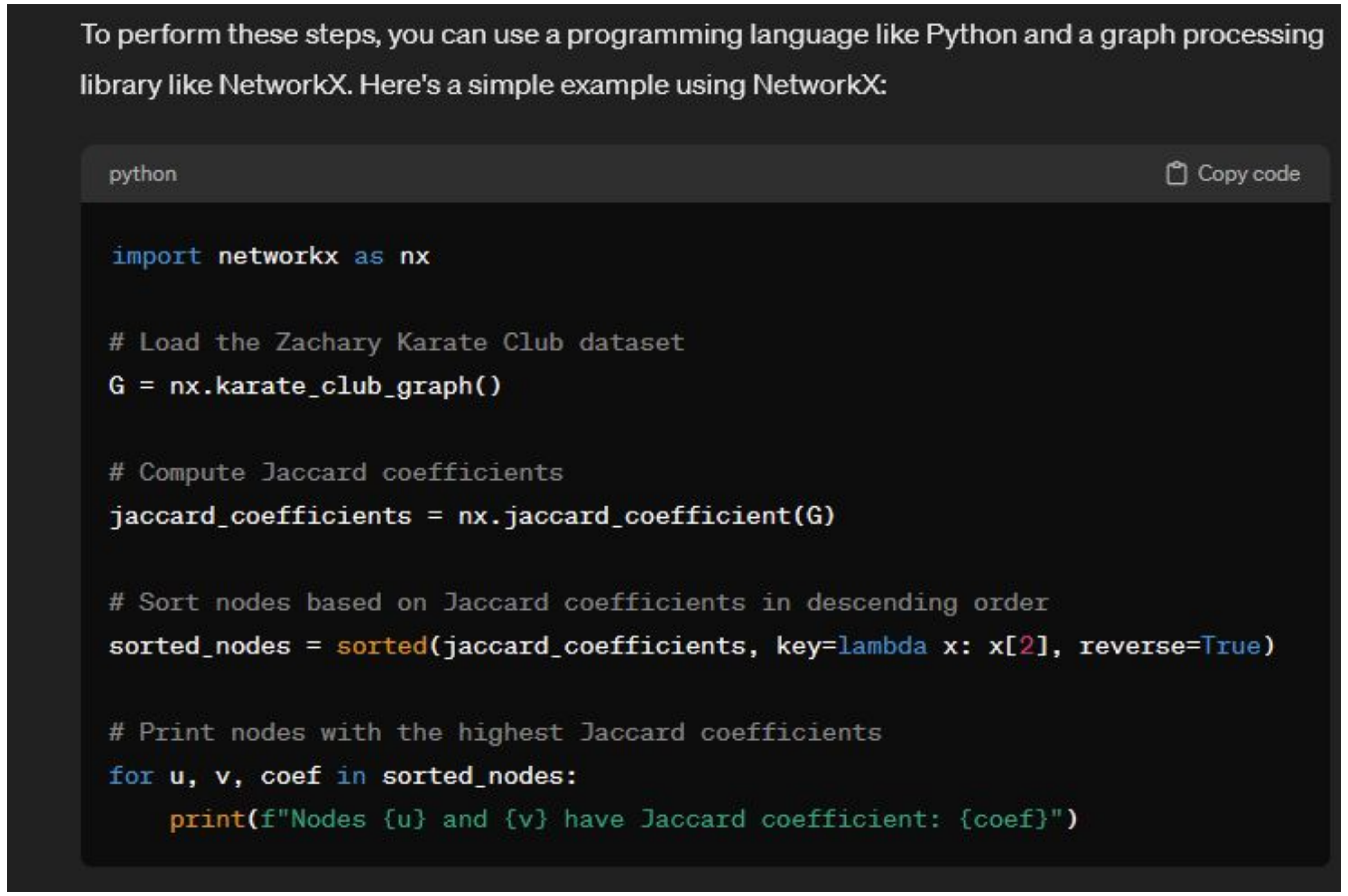
NetworkX

- NetworkX is arguably the most popular graph tool
  - First release was 19 years ago
  - Huge number of graph algorithms and function
  - ~14K stars and 3K forks on GitHub
  - ~47M PyPI and ~360k conda downloads per month

- NetworkX is so popular that even ChatGPT recommends it







Accelerated NetworkX

- NetworkX is arguably
  - First release was 19
  - Huge number of gra
  - ~14K stars and 3K f
  - ~47M PyPI and ~36(
- NetworkX is so popula

How do I
graph?

To find the betweenness centrality va
library in Python. Betweenness centra
along the shortest path between two

Step 1: Install `networkx`

If you haven't already installed `netw

Note: If you change the query to include “GPU” then cuGraph is recommended

how do I find betweenness centrality values for each node in a graph using a GPU?

python Copy code

```python
import cudf
import cugraph

# Convert to cuDF DataFrame
edges_cudf = cudf.DataFrame.from_records(edges.values)

# Create the graph
G = cugraph.Graph()
G.from_cudf_edgelist(edges_cudf, source='src', destination='dst')
```

python Copy code

```python
betweenness = cugraph.betweenness_centrality(G)

# Convert the result to a Pandas DataFrame for easier interpretation
betweenness_df = betweenness.to_pandas()
print(betweenness_df)
```

ı of the Zachary Karate Club dataset have the highest
e answer as node IDs in the graph.

ogramming language like Python and a graph processing
ample using NetworkX:

Copy code

```
ataset

d_coefficient(G)

efficients in descending order
efficients, key=lambda x: x[2], reverse=True)

accard coefficients

ve Jaccard coefficient: {coef}")
```

NetworkX

- NetworkX is arguably the most popular graph tool
  - First release was 19 years ago
  - Huge number of graph algorithms and function
  - ~14K stars and 3K forks on GitHub
  - ~47M PyPI and ~360k conda downloads per month

- NetworkX is so popular that even ChatGPT recommends it

How do I find the betweenness centrality values for each node in a graph?

To find the betweenness centrality values for each node in a graph, you can use the `networkx` library in Python. Betweenness centrality is a measure of the number of times a node acts as a bridge along the shortest path between two other nodes. Here's how you can calculate it:

Step 1: Install `networkx`

If you haven't already installed `networkx`, you can do so using pip:

You

What nodes in a Graph representation of the Zachary Karate Club dataset have the highest jaccard coefficient? Please provide the answer as node IDs in the graph.

To perform these steps, you can use a programming language like Python and a graph processing library like NetworkX. Here's a simple example using NetworkX:

python Copy code

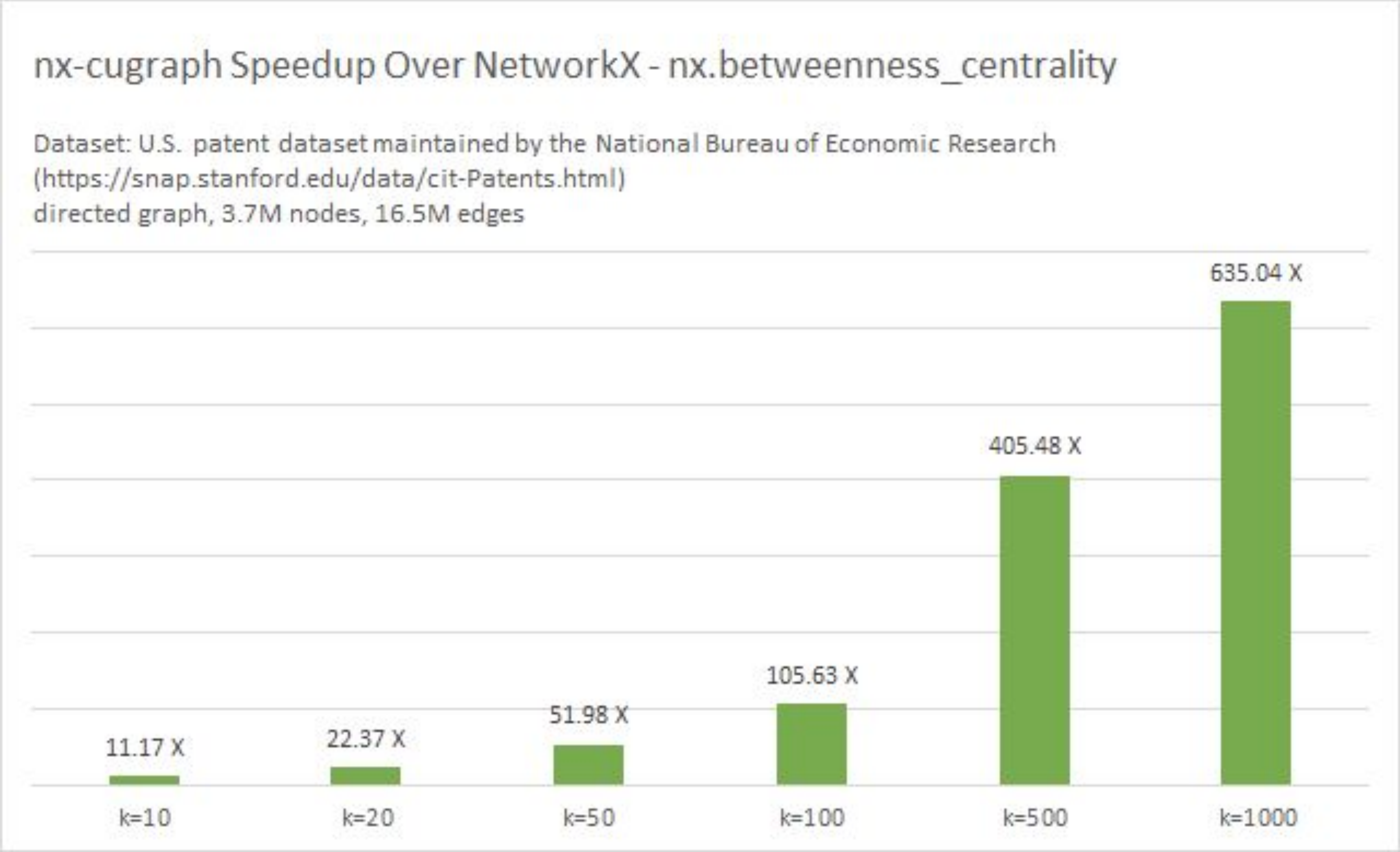
```python
import networkx as nx

# Load the Zachary Karate Club dataset
G = nx.karate_club_graph()

# Compute Jaccard coefficients
jaccard_coefficients = nx.jaccard_coefficient(G)

# Sort nodes based on Jaccard coefficients in descending order
sorted_nodes = sorted(jaccard_coefficients, key=lambda x: x[2], reverse=True)

# Print nodes with the highest Jaccard coefficients
for u, v, coef in sorted_nodes:
    print(f"Nodes {u} and {v} have Jaccard coefficient: {coef}")
```

- We have promoted cuGraph as being “NetwotkX-like” but were never able to make cuGraph a drop-in replacement
- So we shift to Accelerating NetworkX via a cuGraph Backend

Accelerated NetworkX

nx-cugraph: zero-code-change acceleration for NetworkX, powered by cuGraph

- Zero-code-change GPU-acceleration for NetworkX code
- Accelerates up to 600x depending on algorithm and graph size
- Support for 60 popular graph algorithms and growing
- Fallback to CPU for any unsupported algorithms



NetworkX 3.2, CPU: Intel(R) Xeon(R) Platinum 8480CL 2TB, GPU: NVIDIA H100 80GB

```
import pandas as pd
import networkx as nx

url = "https://data.rapids.ai/cugraph/datasets/cit-Patents.csv"
df = pd.read_csv(url, sep=" ", names=["src", "dst"], dtype="int32")
G = nx.from_pandas_edgelist(df, source="src", target="dst")

%time result = nx.betweenness_centrality(G, k=10)
```

```
user@machine:/# ipython bc_demo.ipy
CPU times: user 7min 38s, sys: 5.6 s, total: 7min 44s
Wall time: 7min 44s

user@machine:/# NETWORKX_BACKEND_PRIORITY=cugraph ipython bc_demo.ipy
CPU times: user 18.4 s, sys: 1.44 s, total: 19.9 s
Wall time: 20 s
```

Run on GPU if available, fallback to CPU if not

Simply install nx-cugraph and set environment
> export **NETWORKX_AUTOMATIC_BACKENDS="cugraph"**

nx-cugraph – supported algorithms

centrality
betweenness_centrality
edge_betweenness_centrality
degree_centrality
in_degree_centrality
out_degree_centrality
eigenvector_centrality
katz_centrality

cluster
average_clustering
clustering
transitivity
triangles

community
louvain_communities

components
connected_components
is_connected
node_connected_component
number_connected_components
is_weakly_connected
number_weakly_connected_components
weakly_connected_components

core
core_number
k_truss

dag
ancestors
descendants

isolate
is_isolate
isolates
number_of_isolates

link_analysis
hits
pagerank

operators
complement
reverse

reciprocity
overall_reciprocity
reciprocity

shortest_paths
has_path
shortest_path
shortest_path_length
all_pairs_shortest_path
all_pairs_shortest_path_length
bidirectional_shortest_path
single_source_shortest_path
single_source_shortest_path_length
single_target_shortest_path
single_target_shortest_path_length
all_pairs_bellman_ford_path
all_pairs_bellman_ford_path_length
bellman_ford_path
bellman_ford_path_length
single_source_bellman_ford
single_source_bellman_ford_path
single_source_bellman_ford_path_length

traversal
bfs_edges
bfs_layers
bfs_predecessors
bfs_successors
bfs_tree
descendants_at_distance
generic_bfs_edges

tree
is_arborescence
is_branching
is_forest
is_tree

60 graph algorithms
42 accelerated graph generators (not shown)
More added with every release

NOTE: nx-cugraph is limited to Single GPU

Understand the Algorithms

- Fake Determinism
  - Modularity-based clustering: Louvain, Ledien
    - Given a graph in the same order will produce a similar answer, but re-order the data and you could get a different answer

- Ranking vs Scoring
  - Centrality Algorithms, like Betweenness, produce a score that can be compared across graphs
  - PageRank produces a Ranking. The values cannot be compared across geraphs
  - Modularity is a unitless value that cannot be compared across graphs

- Path Finding (BFS / SSSP) only returns one path
  - If there are two shortest paths of the same length, the algorithm only returns the first one

- Parallel Processing vs Single Thread
  - Order in the returned DataFrame is not guaranteed
  - Least significant bits can fluctuate over runs

Creating a Graph

- Creating a Graph is straight forwards – using cuGraph (creating via pylibcugraph is different)
  - Load the data into either
    - A Pandas DataFrame
    - A cuDF DataFrame
    - A DASK cuDF DataFrame

```
df  = cudf.read_csv(input_data_path, names=['src', 'dst',])
```

  - Create a Graph from the data

```
G = cugraph.Graph()
G.from_cudf_edgelist(df, source=['src'], destination=['dst'])
```

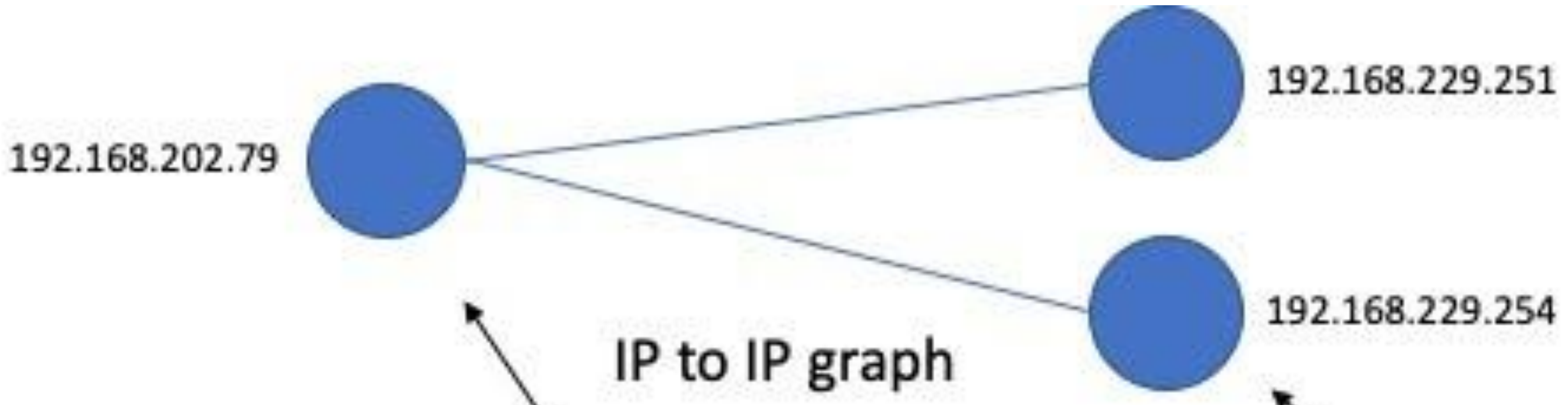
- Now you can run algorithms

```
pr = cugraph.pagerank(G)
```

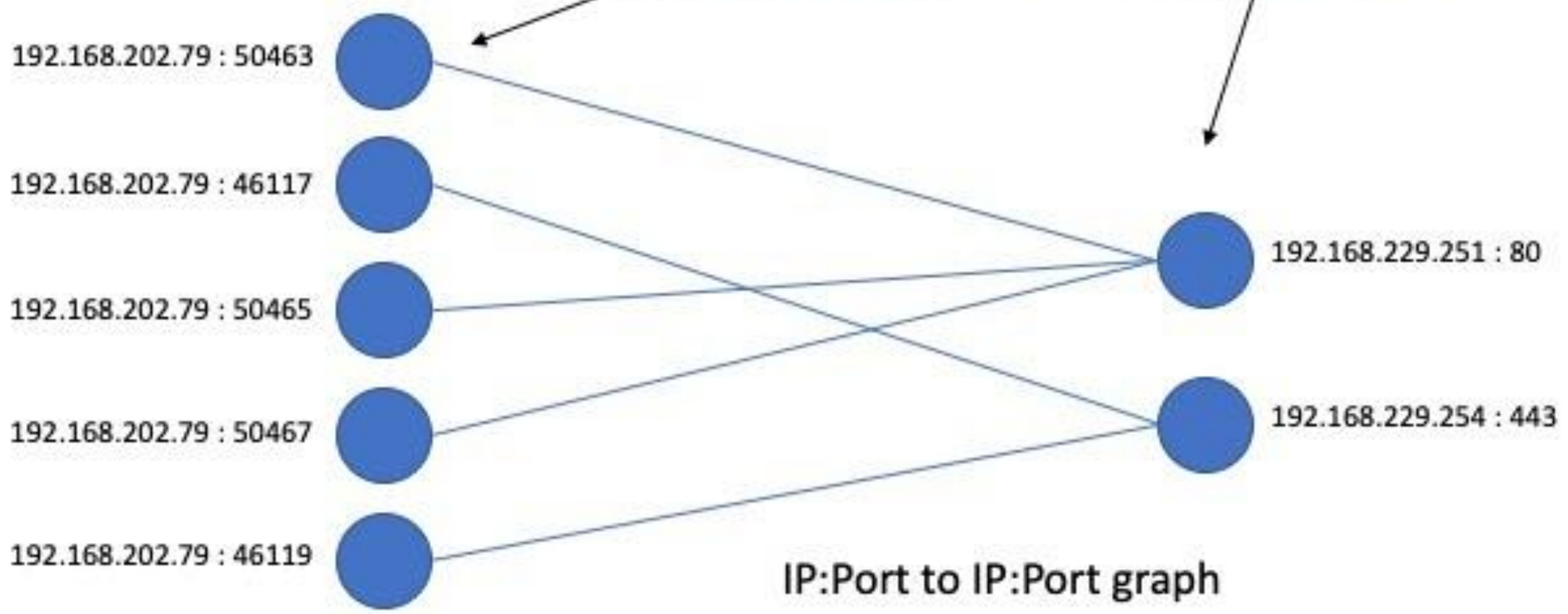
- How does changing what is the "source" and "destination" node affect that graph?

Example with Cyber Daya

Same data – Two different view



ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	resp_bytes
1331901000	CCUIP21wTjqkj8ZqX5	192.168.202.79	50463	192.168.229.251	80	tcp	-	-	-	-
1331901000	Csssjd3tX0yOTPDpng	192.168.202.79	46117	192.168.229.254	443	tcp	-	-	-	-
1331901000	CHEt7z3AzG4gyCNgci	192.168.202.79	50465	192.168.229.251	80	tcp	http	0.01	166	214
1331901000	CKnDAp2ohlvN6rpiXl	192.168.202.79	50467	192.168.229.251	80	tcp	http	0.01	166	214
1331901000	CGUBcoXKxBE8gTNI	192.168.202.79	46119	192.168.229.254	443	tcp	ssl	0.02	544	1060

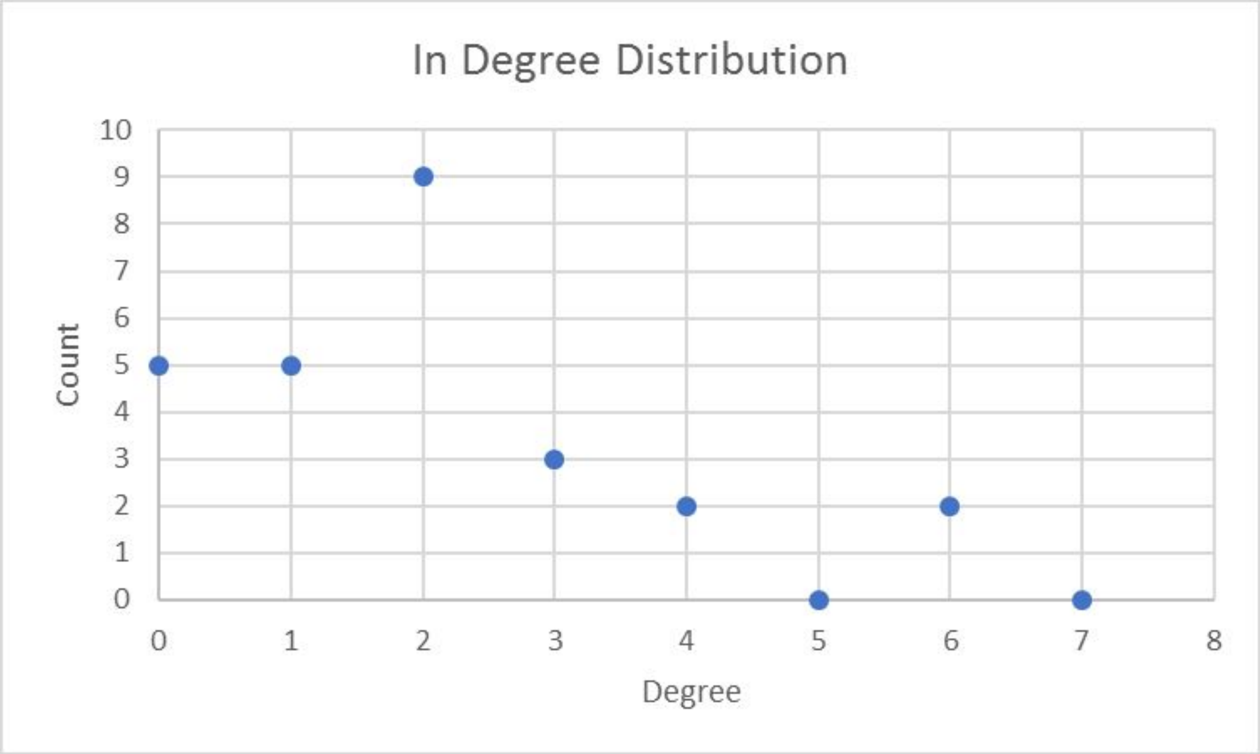


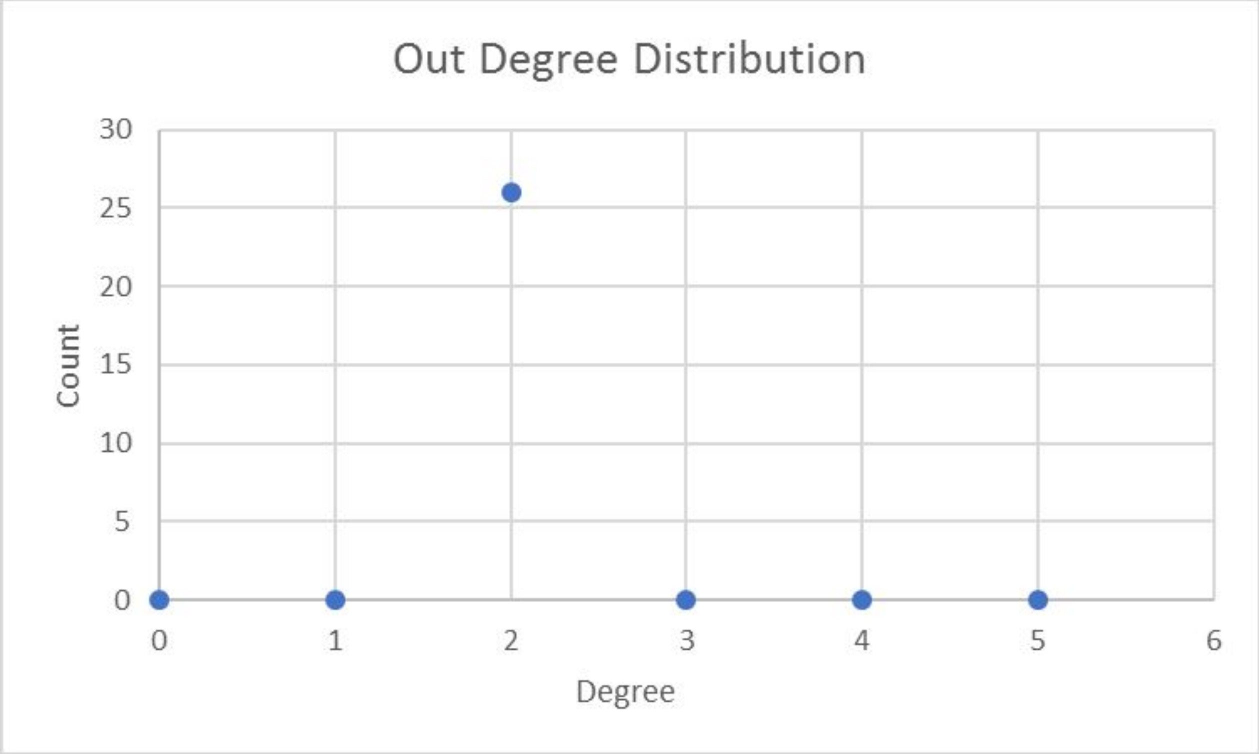
For non-cyber folks: Ports are used for different services

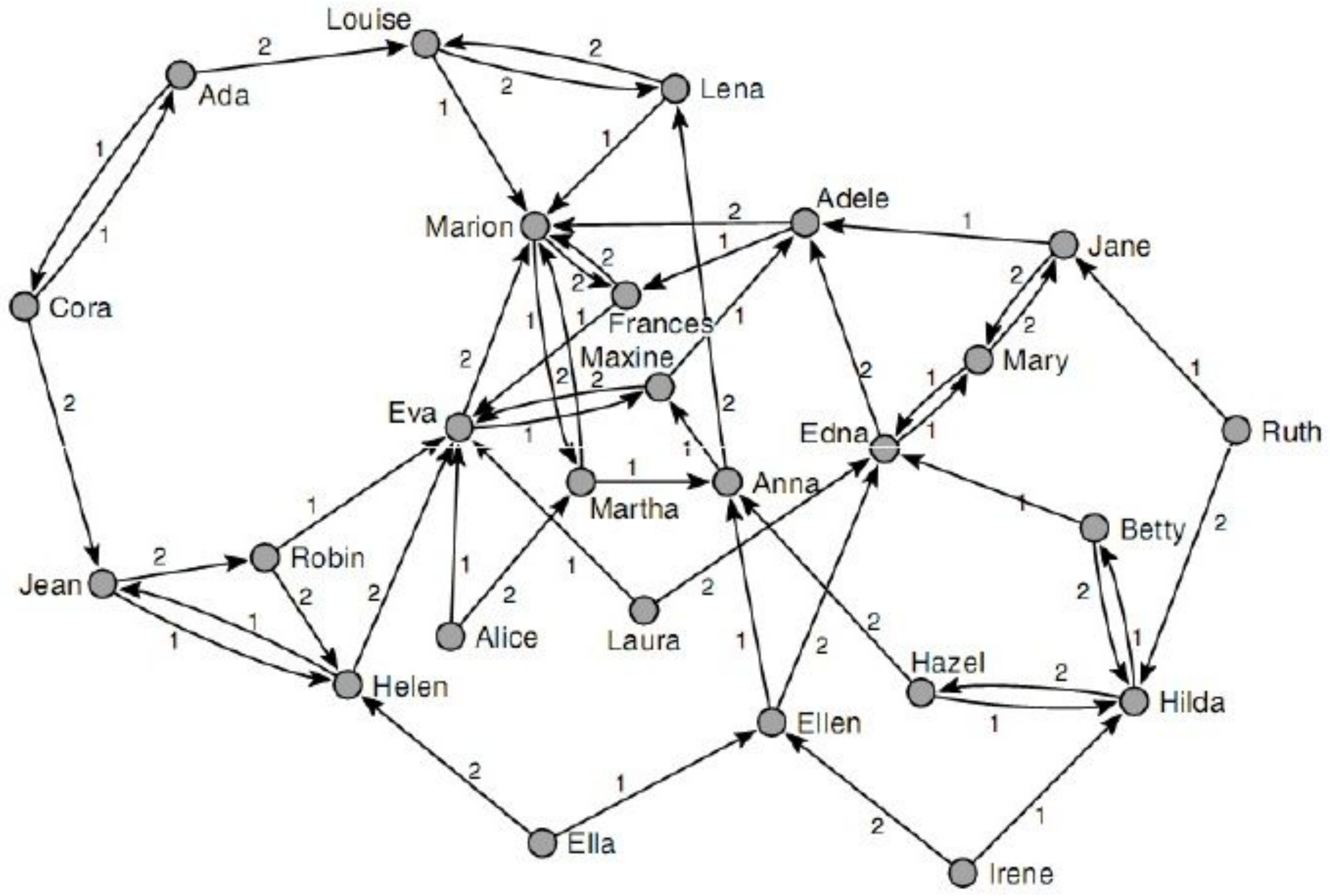
Why Doesn’t My Data Follow a Power Law?

- You need to know where the data comes from and if there are constraints on the data.

- Is your data collection limiting values?







Dining-table partners in a dormitory at a New York State Training School

Image: de Nooy, W., Mrvar, A., and Batagelj, V. Exploratory Social Network Analysis with Pajek. Cambridge University Press, Cambridge, 2005.

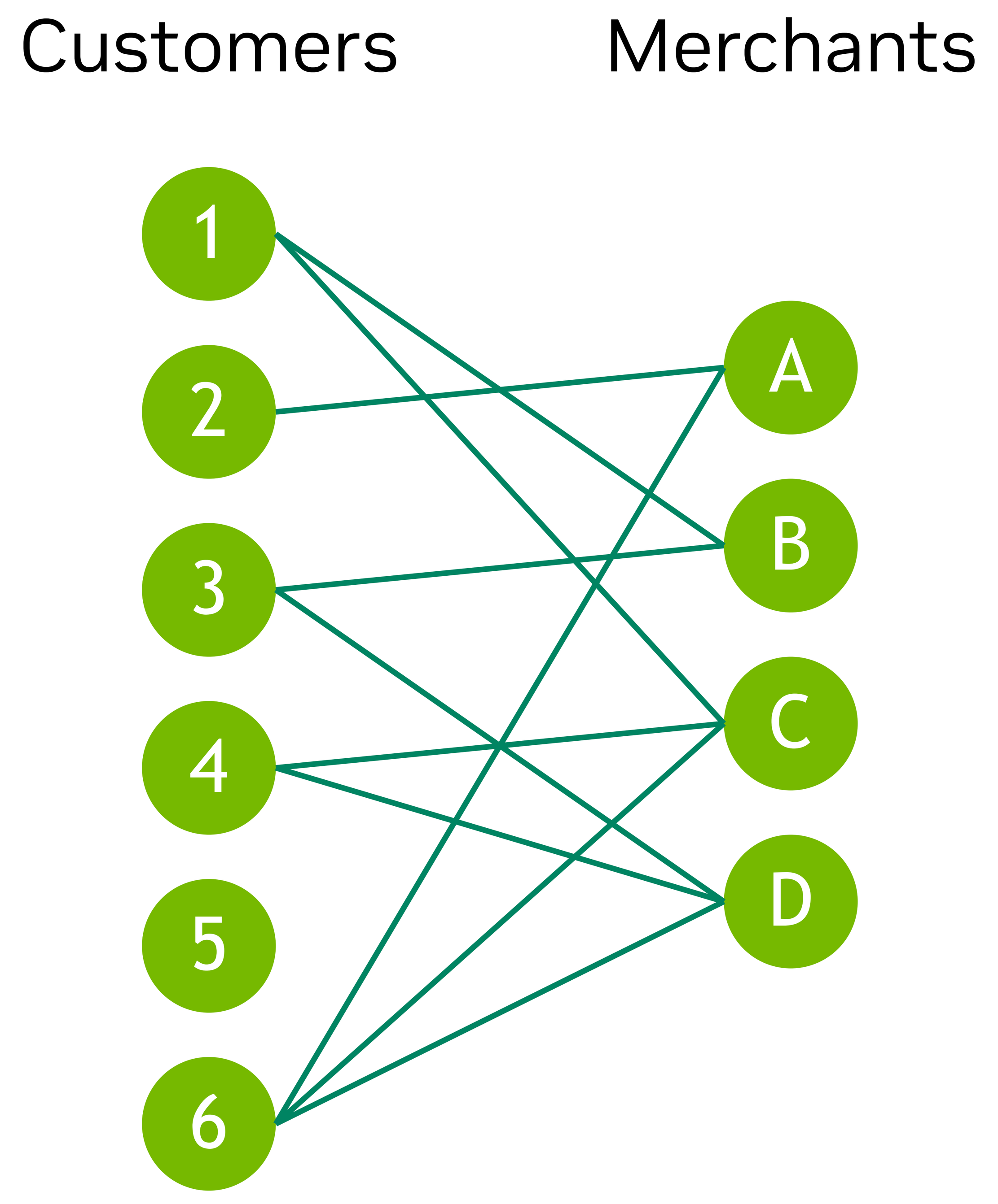
Simple BiPartite Graph

- Great for
  - Recommendation
  - Finding common activity
  - Fraud

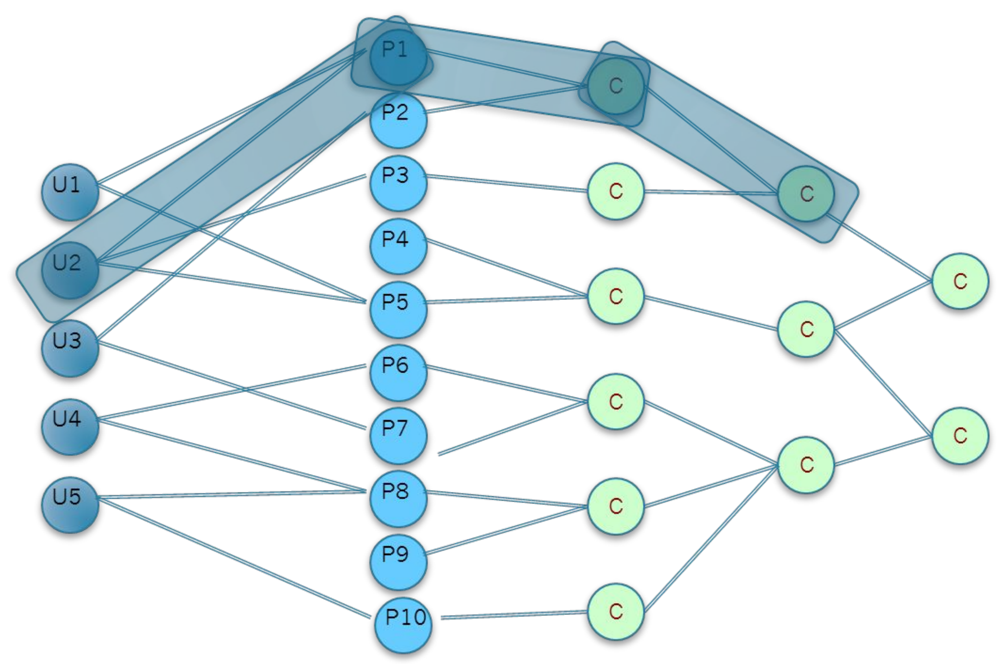
- But
  - PageRank will not work
    - It runs but answers are wrong
  - Louvain / Ledien clustering will not work
    - There is a version called bi-Louvain for bipartite graphs (not not in cuGraph)
  - Triangle Counting will not work
    - There are no triangles

- Moreover, information is lost
  - A customer with multiple cards is lost
  - A merchant with multiple MCCs is lost
  - Note: there is the concept of an N-partite graph

- The Graph is useful for a select set of questions but should not be used for everything

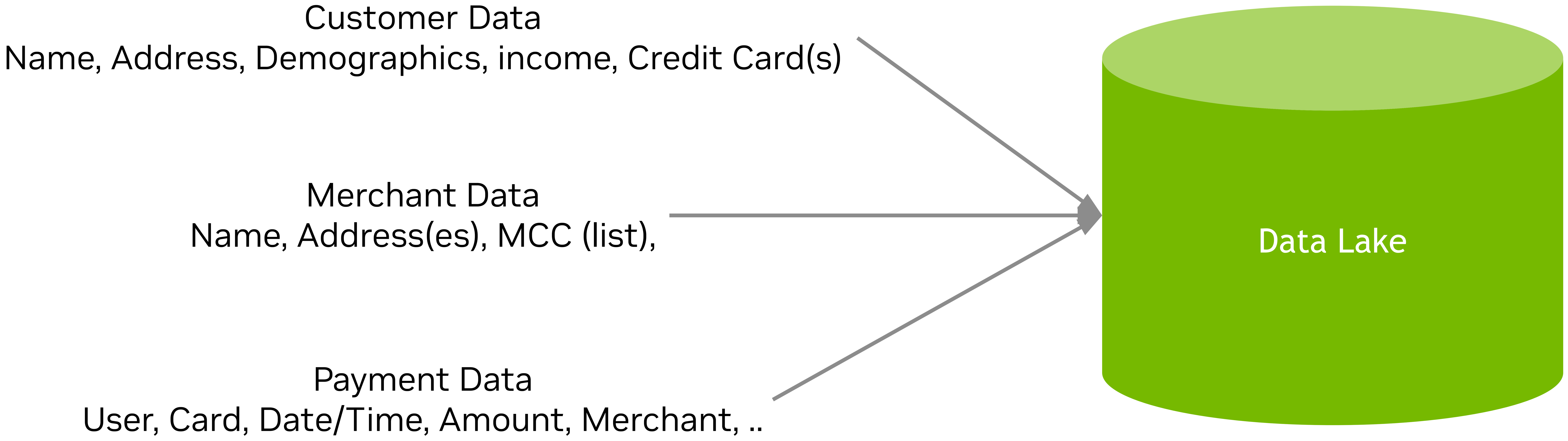


Edges are Transactions



FSI Data

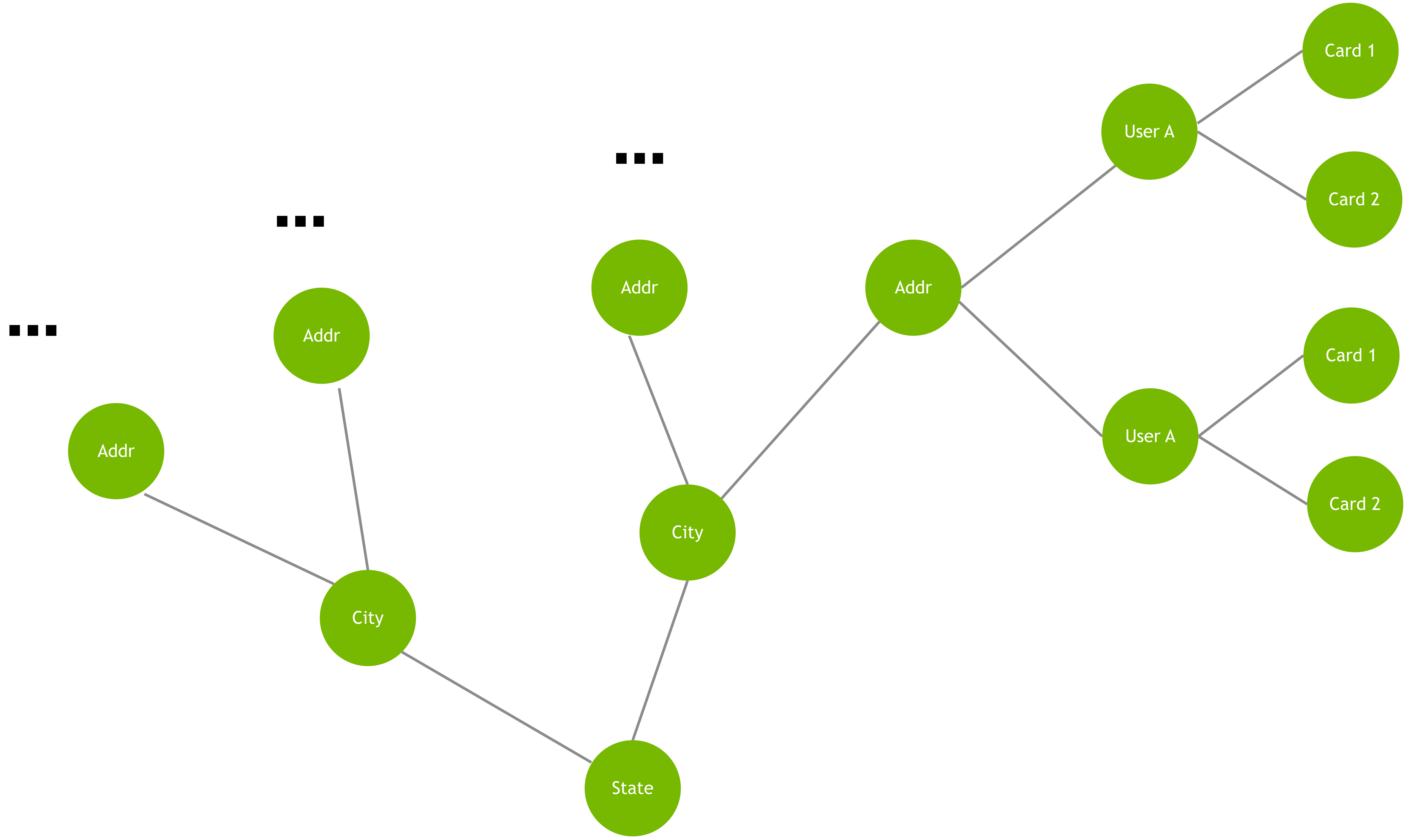
- For this discussion – which might not match reality – let's assume that your data includes



A Reference Graph

- This is a great graph for marketing and analyzing large scale trends.

- But
  - Traversal through City/State nodes does not carry any significance.
  - Mixing node types can produce misleading answers



The IBM TabFormer Dataset

Popular Synthetic Dataset

Apache 2 License

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
0	0	0	2002	9	1	06:21	$134.09	Swipe Transaction	3527213246127876953	La Verne	CA	91750.0	5300	<NA>	No
1	0	0	2002	9	1	06:42	$38.48	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	<NA>	No
2	0	0	2002	9	2	06:22	$120.34	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	<NA>	No

- The data does need to be cleaned up a little
  - I like "Card" to be a unique 12-digit number, for example
  - And processing strings is not always the best, or even supported

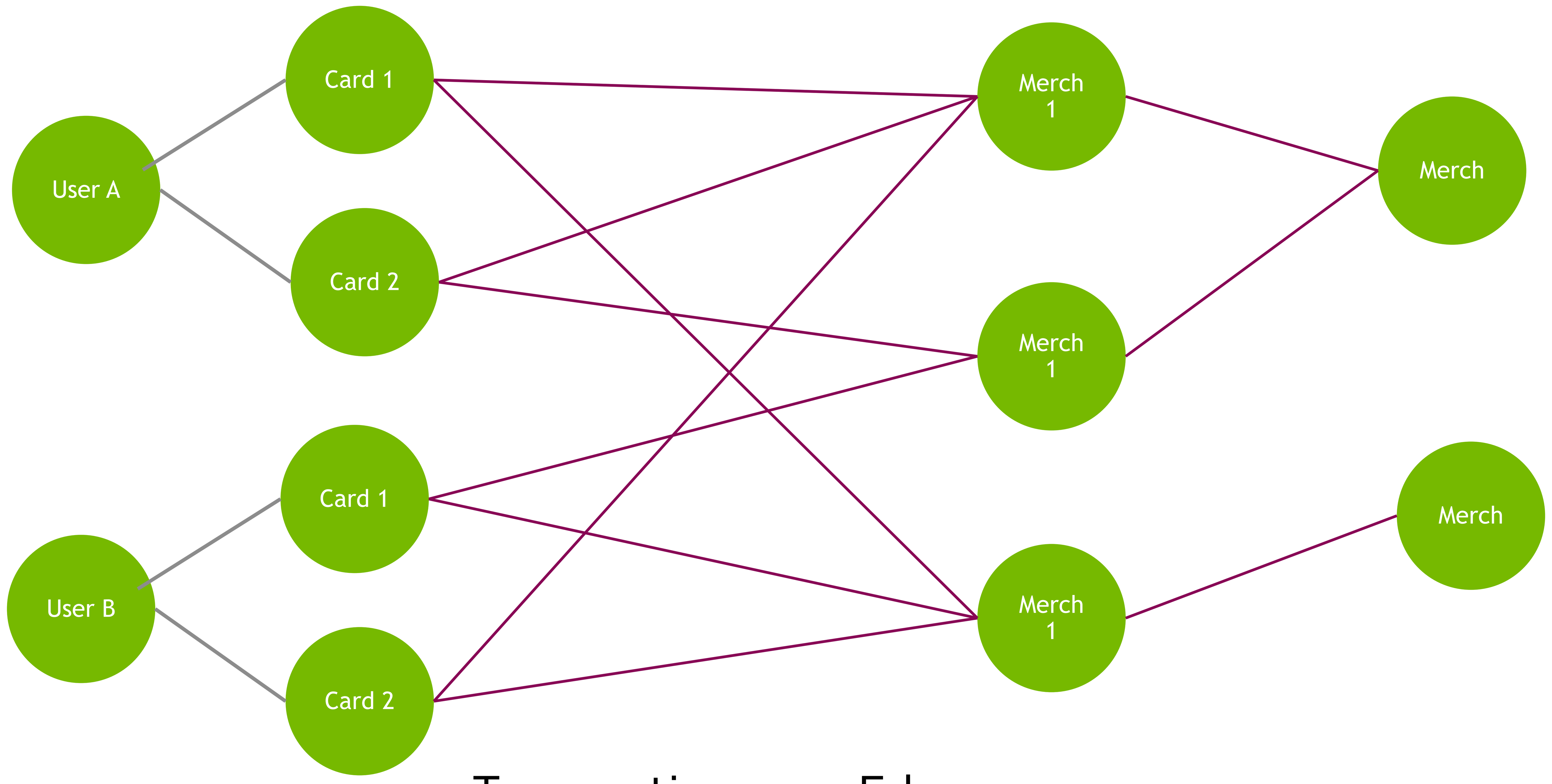
- What question is: How should you form a graph?

https://github.com/IBM/TabFormer

TabFormer:

Wide Graph

- A unique Merchant is “Merchant Name” + “Merchant City” + “Merchant State”
  - You could break a merchant into subparts by including MCC



Transactions are Edges

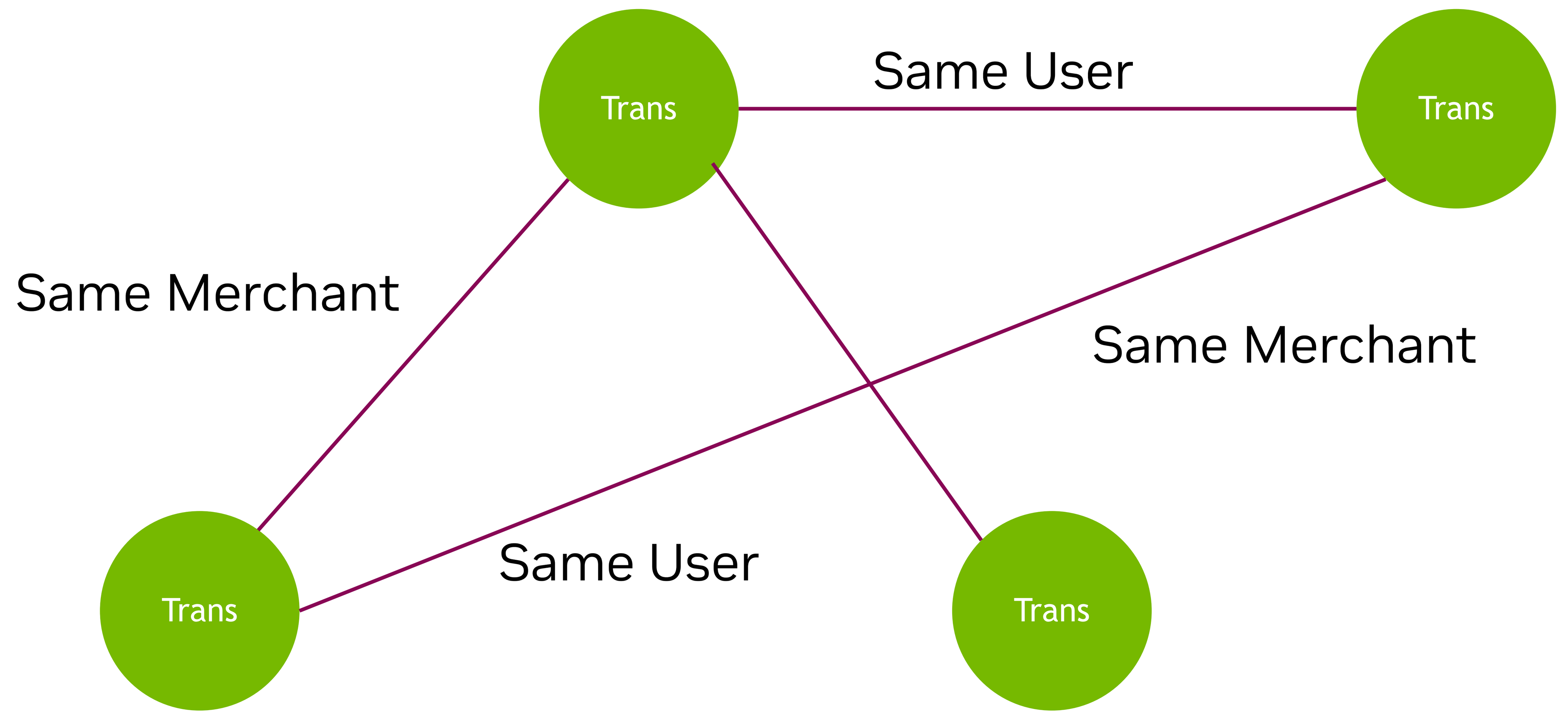
Just using TabFormer data – this graph does not perform well for GNNs since there is no feature associated with the different nodes
But this is a good graph for modeling user behavior

TabFormer

Flipping View

- Transactions are nodes
- Edges are common Users or Merchants between Transactions

- You can now run Node-based algorithms to score Transactions



This is also called a Line Graph (https://en.wikipedia.org/wiki/Line_graph)

Thank You

Questions?