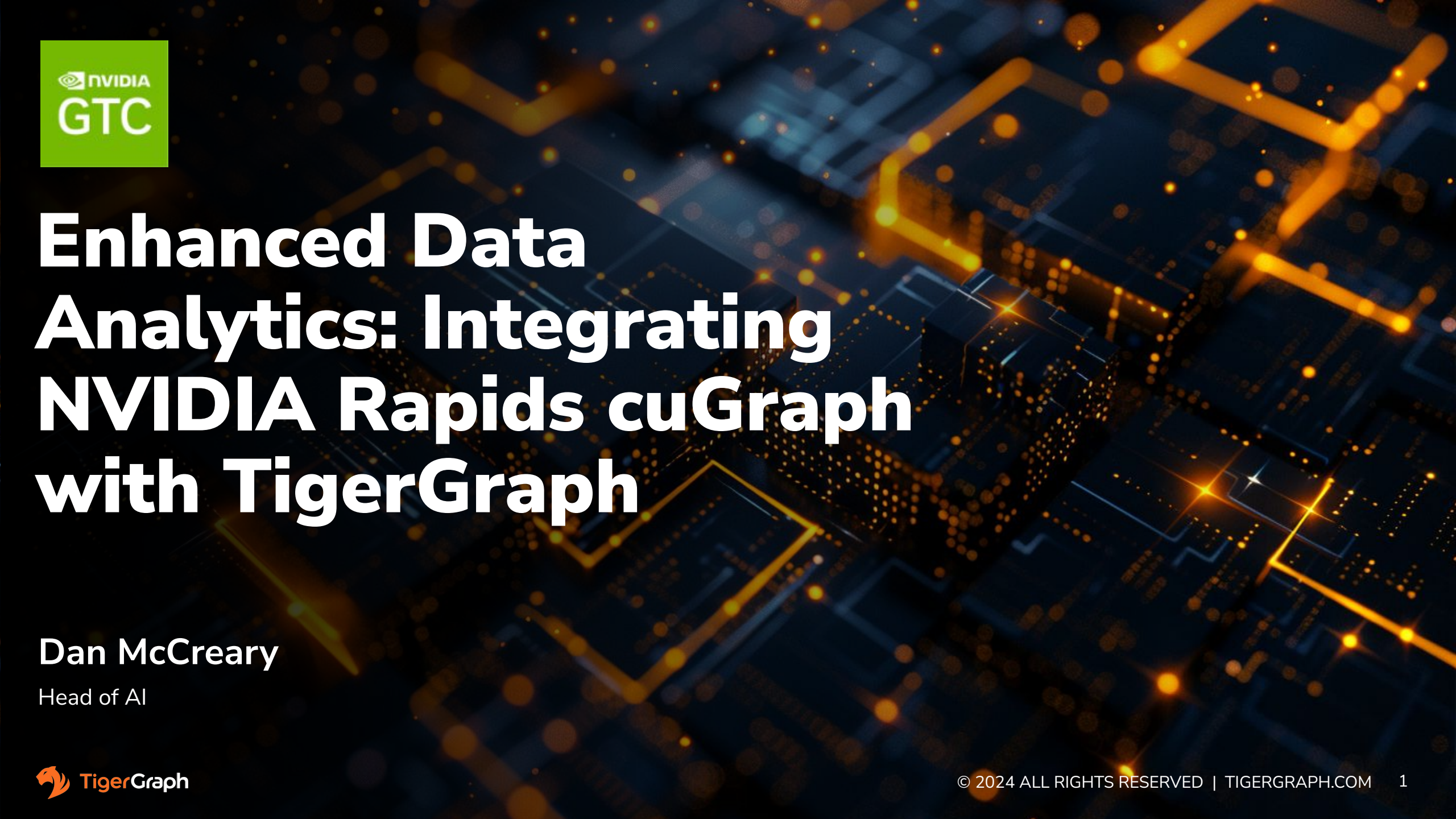



# Enhanced Data Analytics: Integrating NVIDIA Rapids cuGraph with TigerGraph

Dan McCreary

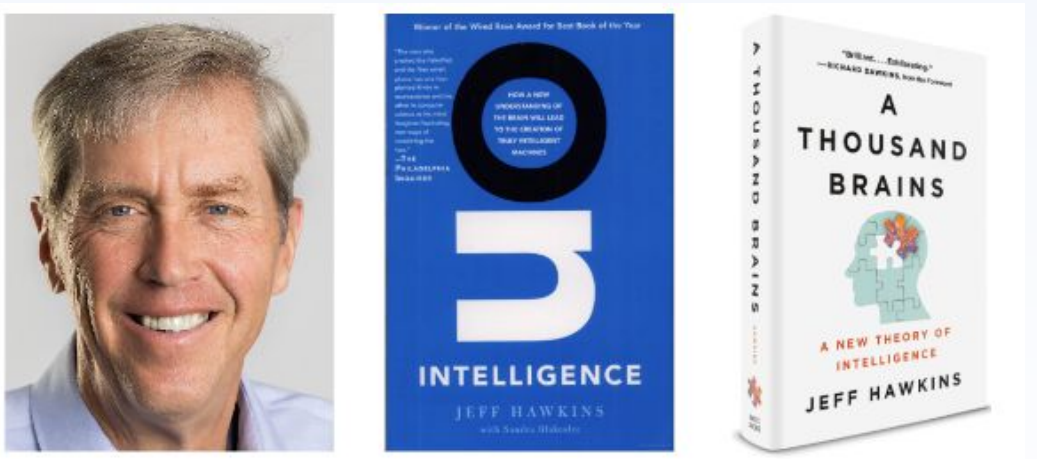
Head of AI



# Jeff Hawkins on AI

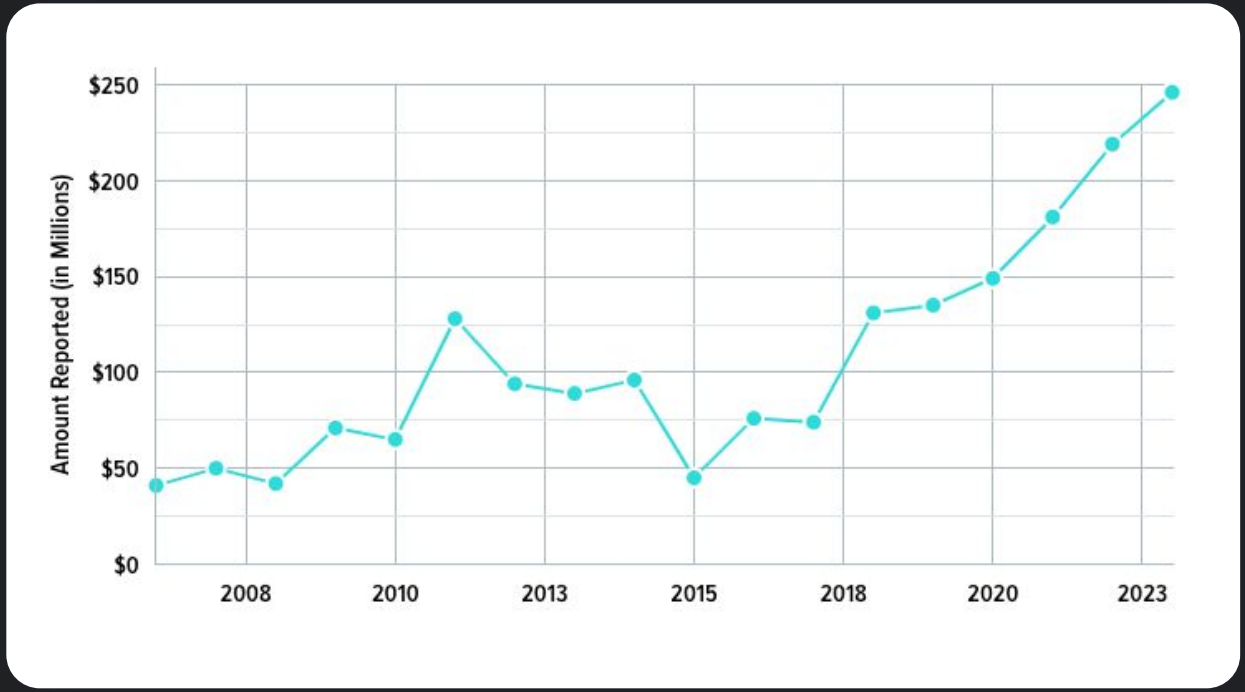
> The key to artificial intelligence has always been the **representation**.
>
> — **Jeff Hawkins on AI**



# Sample Challenge: Banking Transaction Fraud

- Have you ever wondered what happens when you use your credit card at a retailer?
- The authorization usually is returned in just a **few seconds** (Service Level)
- Did you know that many banks execute tens of thousands of checks for each transactions to make sure the transaction is not fraudulent?
- But are these checks enough?

# Credit Card Fraud Continues to Grow




According to WalletHub, the total worldwide credit card fraud rates continue to rise every year.

Annual global fraud losses (credit & debit card): **$34.36** billion (2022)

The total value of fraud  in the US soared to $246 million in 2023, signifying a substantial 12% rise from the previous year.

https://wallethub.com/edu/cc/credit-card-fraud-statistics/25725

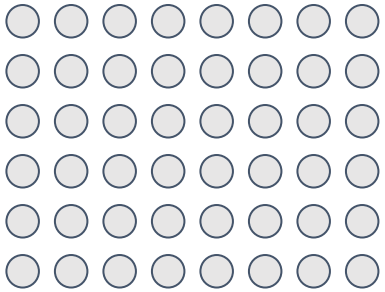
# Introduction

## Dan McCreary

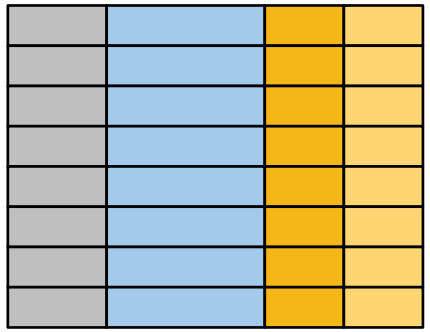
- Head of AI for TigerGraph you use your credit card at a retailer?
- TigerGraph is a market leader in scale-out enterprise graph databases
- We combine LLMs with scale-out graph representations of the world
- Large market share of financial institutions doing fraud detection and anti-money laundering
- Proponent of “Hardware Optimized Graph” (HOG)

# The “Big Four” Representations in AI Today

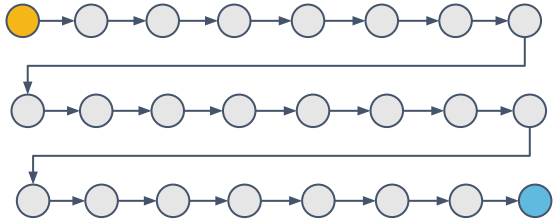
## IMAGE AND VIDEO



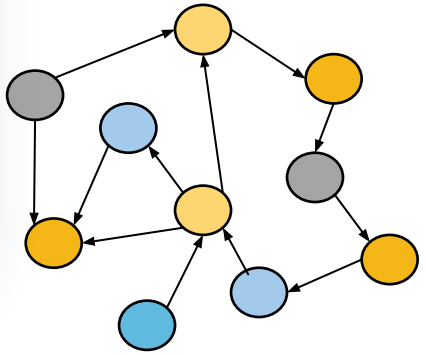
## TABULAR



## SEQUENCES



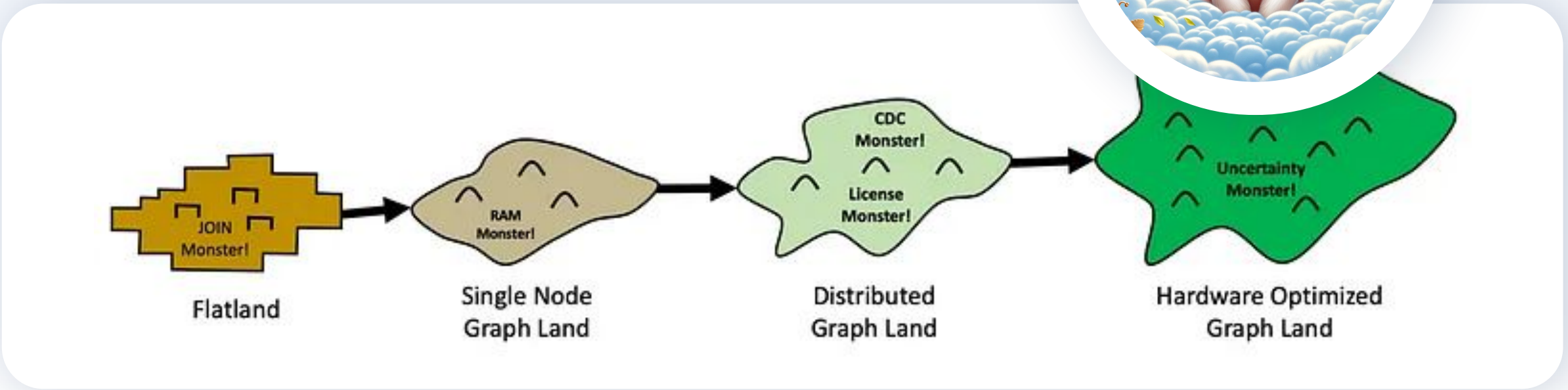
## GRAPH



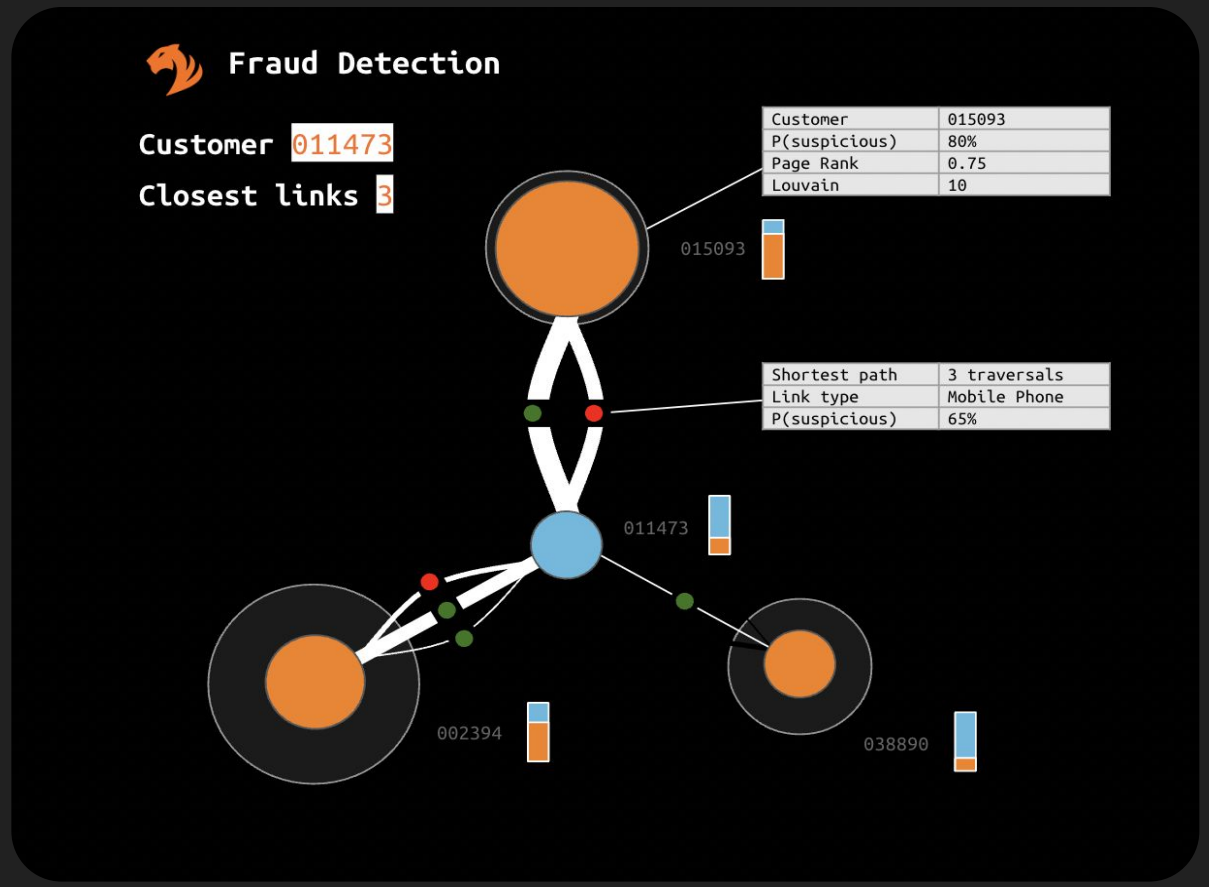
? Which is most similar to the brain?

# Our Journey: Hardware Optimized Graph (HOG Heaven)

https://towardsdatascience.com/from-flatland-to-hog-heaven-the-four-lands-of-ekg-adoption-945571c09b67



# Background on Graph Acceleration




1. Many companies have requirements for mission critical real-time graph algorithms.

   **Example:** is this credit-card transaction fraudulent?

   **Example:** Is this transaction part of a money laundering scheme?

2. Service Level Agreements (SLAs) need to allow algorithms to run in under 100 milliseconds

3. Most CPUs don't offer enough parallelism to execute complex graph algorithms within this SLA window

# Some Terms

**GPU** - Graphics Processing Unit - optimized for matrix manipulation - they typically come with thousands of cores that run in parallel

**CUDA** - NVIDIA C-language application interface to communicate with GPU

**RAPIDS** - an open source Python library for using GPUs http://rapids.ai/

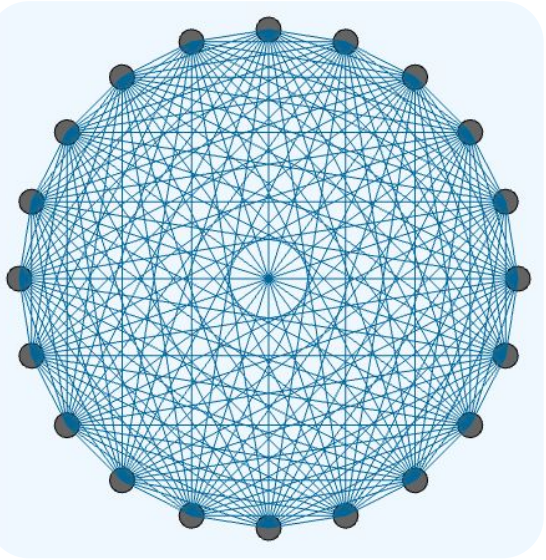
**cuGraph** - A open source collection of GPU accelerated graph algorithms and graph utility functions that are designed to work with Python libraries such as NetworkX

**NVIDIA RAPIDS** - libraries for accelerating the flow of data between CPU and NVIDIA GPUs

**Sparse matrix** - a matrix where most of the values in the matrix are zero - most graph problems have an sparse adjacency matrix

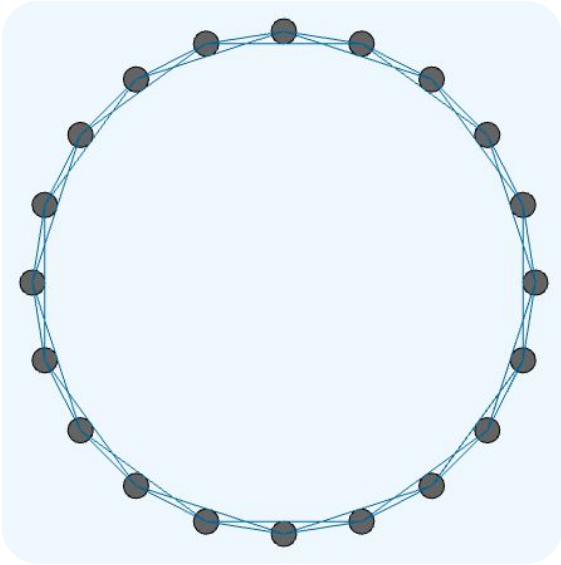
# Dense and Sparse Matrix Representation

## DENSE NETWORK



Everything is connected to everything

## SPARSE NETWORK



Few connections per node (low degree)

GPUs are ideal for **dense** matrix problems

# Dense vs Sparse Matrix

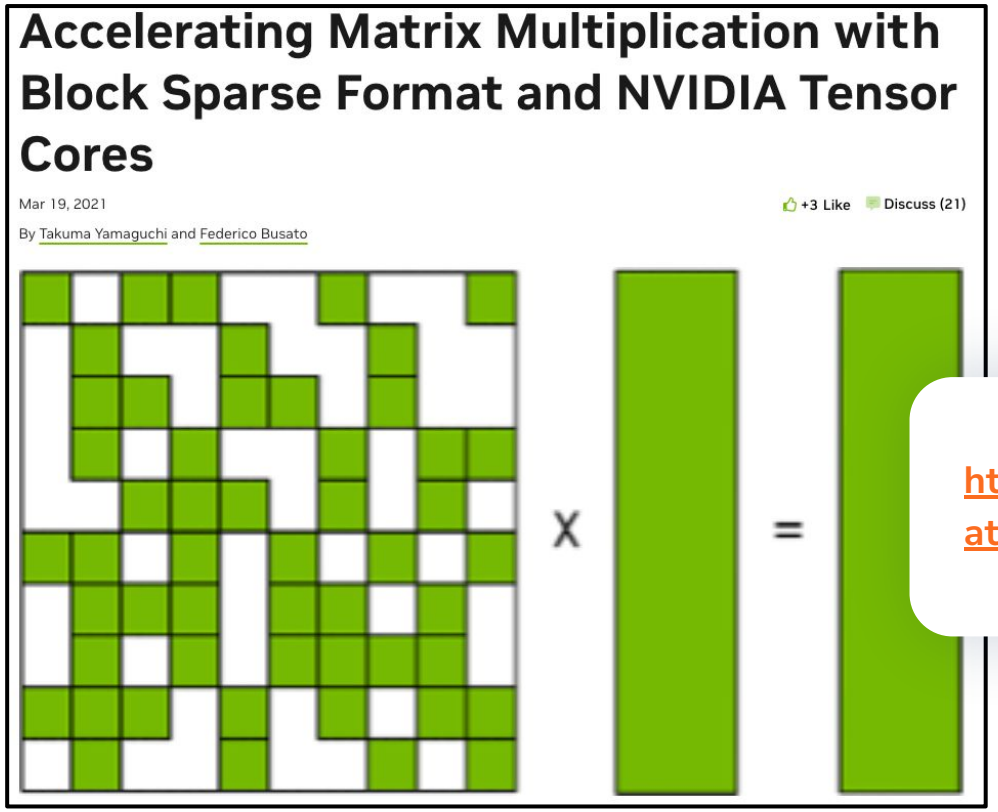
## Dense Matrix

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 31 | 2 | 9 | 7 | 34 | 22 | 11 | 5 |
| 11 | 92 | 4 | 3 | 2 | 2 | 3 | 3 | 2 | 1 |
| 3 | 9 | 13 | 8 | 21 | 17 | 4 | 2 | 1 | 4 |
| 8 | 32 | 1 | 2 | 34 | 18 | 7 | 78 | 10 | 7 |
| 9 | 22 | 3 | 9 | 8 | 71 | 12 | 22 | 17 | 3 |
| 13 | 21 | 21 | 9 | 2 | 47 | 1 | 81 | 21 | 9 |
| 21 | 12 | 53 | 12 | 91 | 24 | 81 | 8 | 91 | 2 |
| 61 | 8 | 33 | 82 | 19 | 87 | 16 | 3 | 1 | 55 |
| 54 | 4 | 78 | 24 | 18 | 11 | 4 | 2 | 99 | 5 |
| 13 | 22 | 32 | 42 | 9 | 15 | 9 | 22 | 1 | 21 |

## Sparse Matrix

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | . | 3 | . | 9 | . | 3 | . | . | . |
| 11 | . | 4 | . | . | . | . | . | 2 | 1 |
| . | . | 1 | . | . | . | 4 | . | 1 | . |
| 8 | . | . | . | 3 | 1 | . | . | . | . |
| . | . | . | 9 | . | . | 1 | . | 17 | . |
| 13 | 21 | . | 9 | 2 | 47 | 1 | 81 | 21 | 9 |
| . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | 19 | 8 | 16 | . | . | 55 |
| 54 | 4 | . | . | . | 11 | . | . | . | . |
| . | . | 2 | . | . | . | . | 22 | . | 21 |

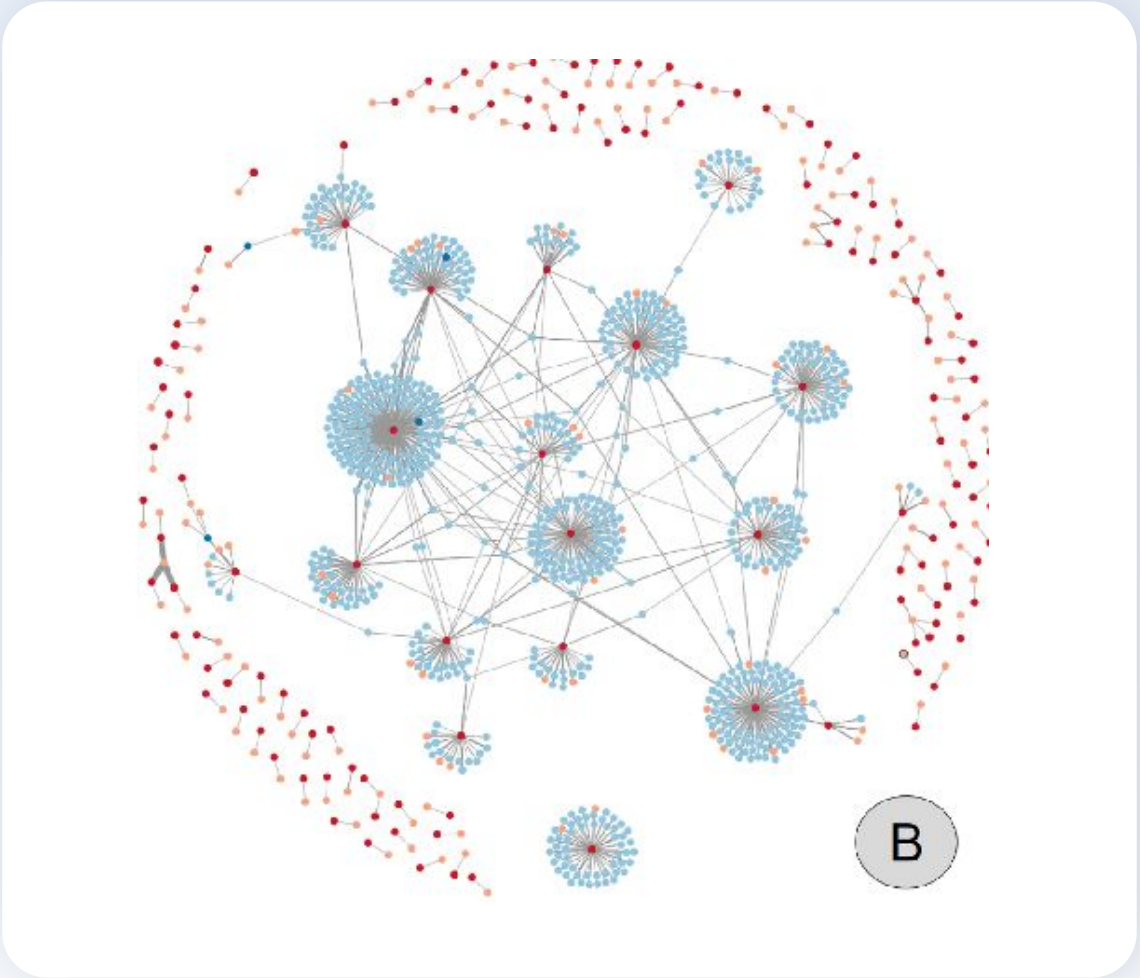
# NVIDIA Library for Sparse Matrix Representations (March 2021)




https://developer.nvidia.com/blog/accelerating-matrix-multiplication-with-block-sparse-format-and-nvidia-tensor-cores/

# Solution!

Leverage the **massive parallelism** of both TigerGraph + NVIDIA GPUs to analyze complex networks in real-time with **cuGraph**



# Accelerating TigerGraph for Enhanced Analytics

**Performance Enhancement:** Accelerating TigerGraph improves processing speed and efficiency for timely insights.

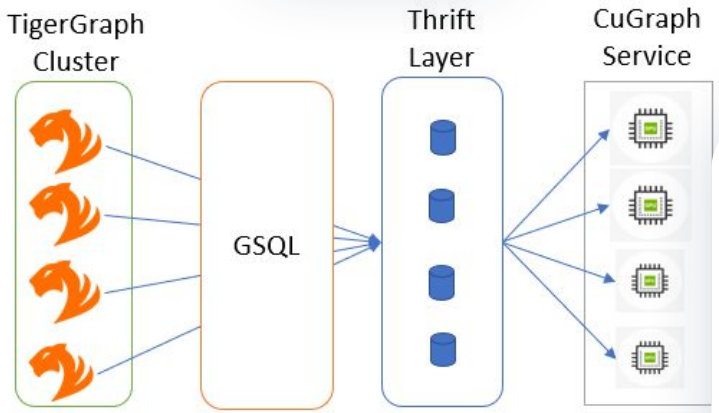
**Scalability:** Acceleration enables TigerGraph to handle larger and more complex datasets seamlessly.

**Competitive Advantage:** Faster insights from accelerated TigerGraph drive business success in a data-driven landscape.

**Enabling Advanced Applications:** Accelerating TigerGraph unlocks the potential of cutting-edge technologies like graph-based machine learning and AI-driven analytics.

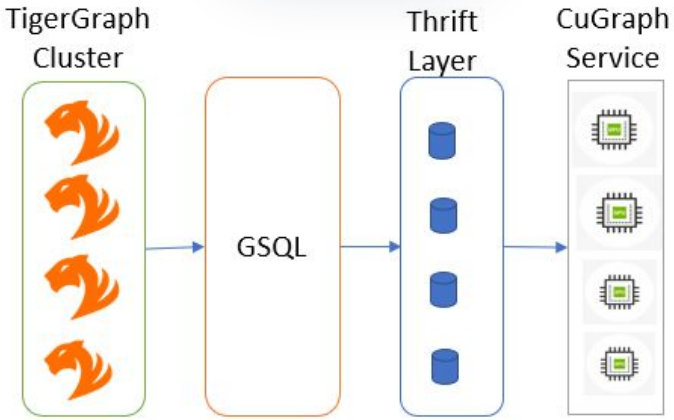
# Initial Approach - TigerGraph integration with cuGraph via Python APIs

**STEP 1**



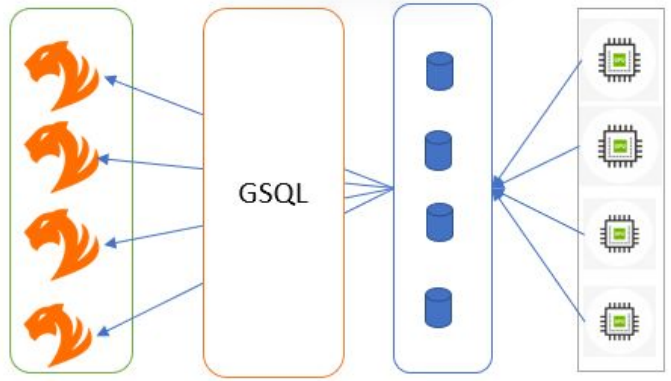

Step 1: Streamlining data flow from TigerGraph nodes to NVIDIA GPUs with Zetta bridge and GSQL

**STEP 2**




Step 2: Invoking GPU-Accelerated Algorithms on Streamed Data with GSQL

**STEP 3**




Step 3: Parallel Data Retrieval from GPU and Storage on TigerGraph as Vertex Attributes

# Evolution from Python to Native C++ cuGraph Integration

## CHALLENGES WITH THE PREVIOUS ARCHITECTURE:

- Dependency on Disk: We are using disk based data transfer between TigerGraph and cuGraph.
- Python Dependency: Utilizing Python for cuGraph introduced overhead and complexity, impacting performance.
- Thrift Layer : Thrift communication added complexity and overhead, potentially affecting system reliability.

## IMPLICATIONS FOR SETUP AND PERFORMANCE:

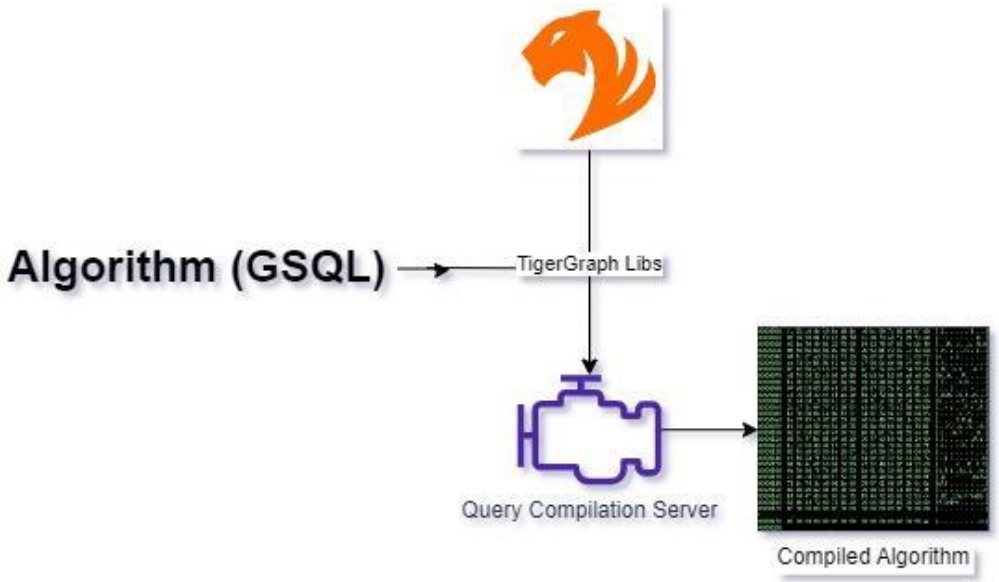
- Complex setup process hindered optimal performance.
- Shared disk, Python, and Thrift dependencies created scalability challenges.

## NEXT-GENERATION ARCHITECTURE:

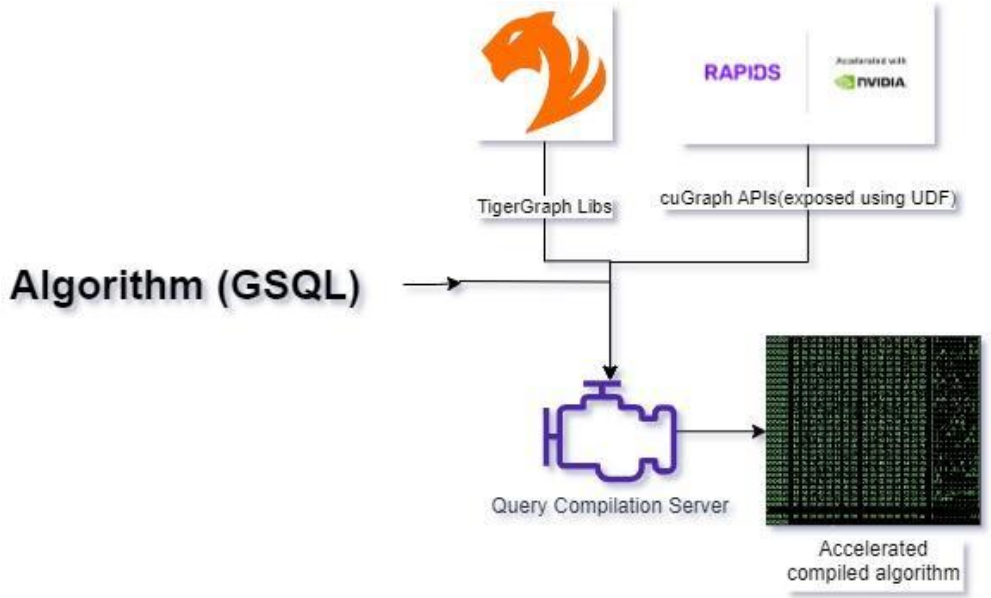
- Designed to overcome limitations of the previous architecture.
- Revolutionizes graph processing for enhanced scalability and performance.
- Explores new frontiers in graph analytics through native C++ cuGraph integration.

# Compilation - Native integration of Rapids with TigerGraph

**REGULAR**

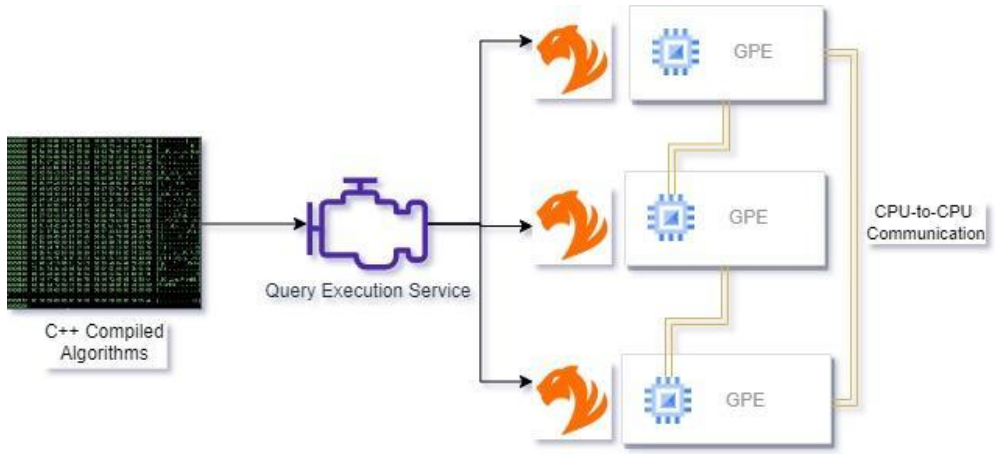



**RAPIDAS CAPABLE SYSTEM**

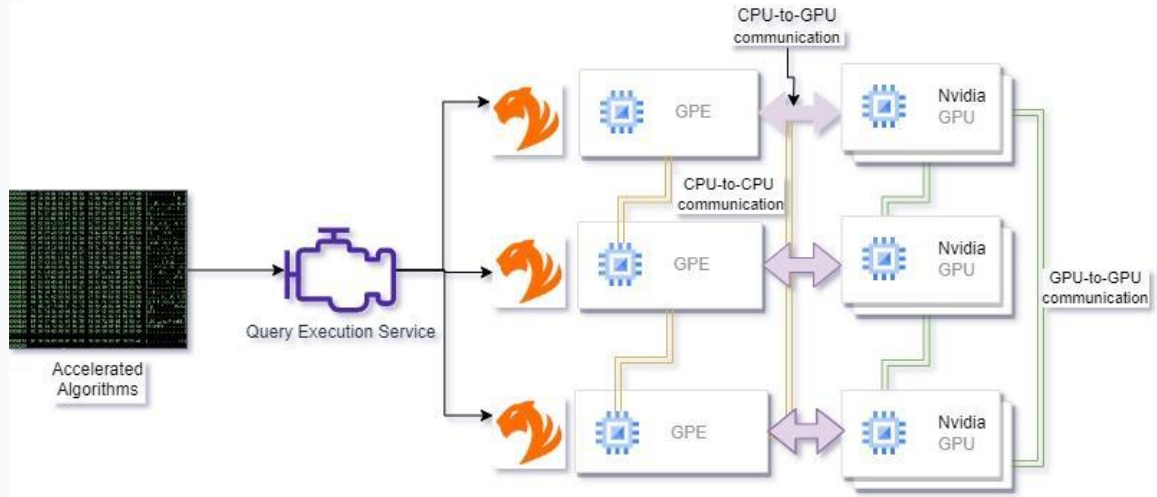


# [Execution] Native integration of Rapids with TigerGraph

**REGULAR CLUSTER**




**RAPIDS CAPABLE SYSTEM**



# Advantages of Next-Gen Architecture

## UNPRECEDENTED PERFORMANCE:

- Leveraging GPU acceleration and streamlined processing pipeline.
- Enables rapid execution of complex graph algorithms.

## SIMPLIFIED SETUP:

- Integration of cuGraph and ZettaAccel into TigerGraph.
- Eliminates complex dependencies, reduces configuration overhead.

## SCALABILITY AND FLEXIBILITY:

- Dynamic GPU resource management.
- Streamlined data transfer mechanisms.
- Effortlessly scales to handle large-scale datasets and diverse workloads.

# Use an Published Graph Benchmark: LDBC

## GRAPH ANALYTICS OVERVIEW:

Developed by the Linked Data Benchmark Council (LDBC).

Comprehensive suite for evaluating graph database management systems and processing frameworks.

## KEY FEATURES:

Real-world datasets and diverse workloads provided.

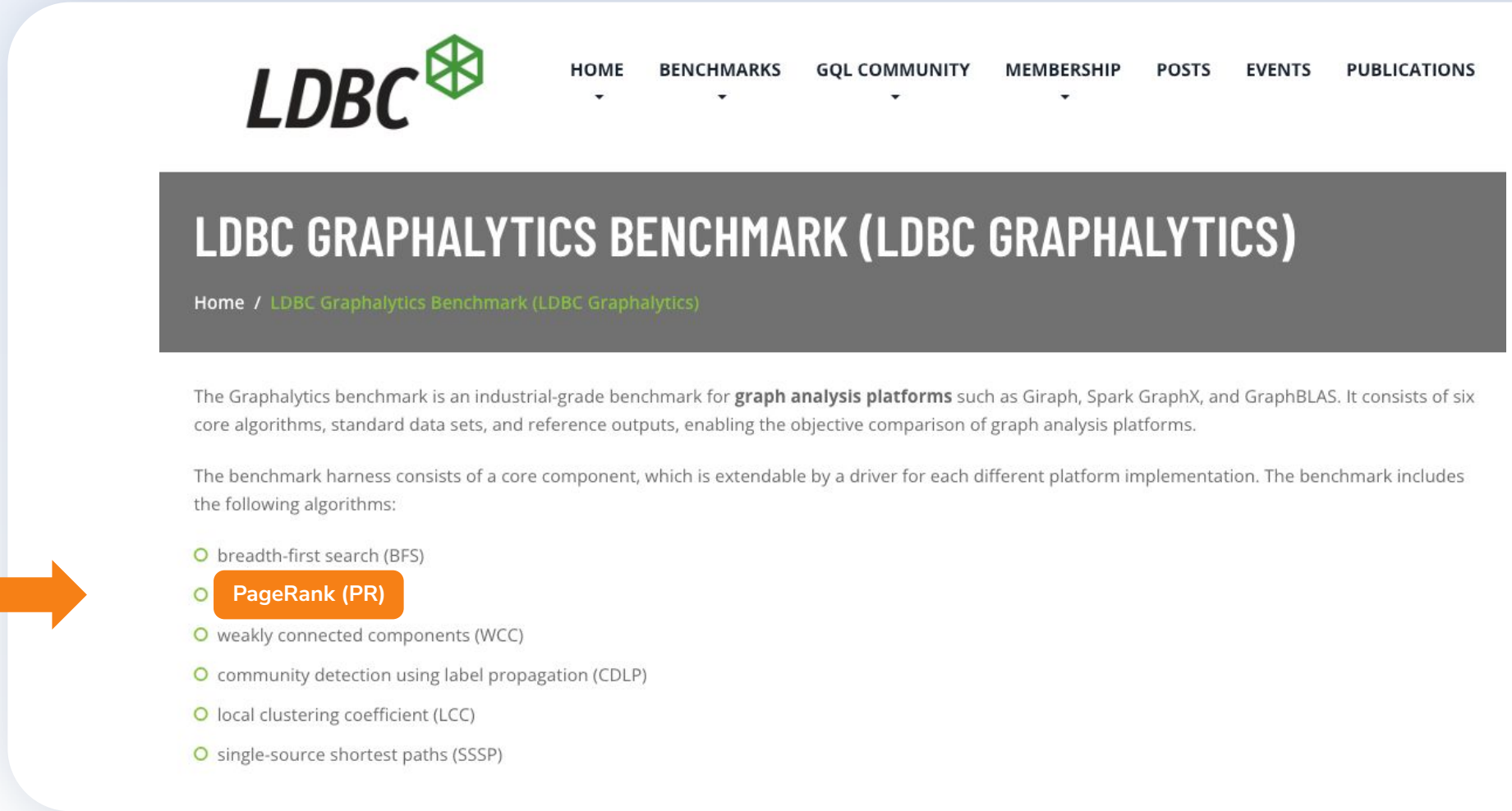
Includes a range of graph algorithms for testing system efficiency and scalability.

## PURPOSE:

Helps researchers and organizations assess graph-related technologies.

Facilitates informed decisions in selecting and optimizing graph databases and processing frameworks.

# LDBC Graphalytics PageRank Benchmark



# PageRank Algorithm Performance

| Scale Exponents of 2 | Vertices | Edges | TigerGraph Cluster (Sec) | cuGraph + TigerGraph (python) (Sec) | cuGraph + TigerGraph (Native) (Sec) |
|---|---|---|---|---|---|
| Graph 22 | 2.39M | 64M | 311.162 | 12.14 (25X) | 6.91 (45X) |
| Graph 23 | 4.6M | 129M | 617.82 | 14.44 (42X) | 9.04 (68X) |
| Graph 24 | 8.87M | 260M | 1205.34 | 24.63 (48X) | 14.69 (82X) |
| Graph 25 | 17.06M | 523M | 2888.74 | 42.5 (67X) | 21.09 (137X) |
| Graph 26 | 32.8M | 1.05B | 4842.4 | 73.84 (65X) | 41.01 (118X) |

**Over 1 Billion Edges!**

| | |
|---|---|
| Total Nodes | 2 Nodes |
| Each node has: | |
| RAM | 512G |
| CPU Cores | 128 Core |
| A100 GPUs | 4 with 40G RAM each |
| Connection | UCX |
| Rapids Version | rapids-23.12 |

Native cuGraph Integration Shows Nearly 2x Performance Boost Over Python, with Over 100X Improvement in Overall Runtime

Note: This is an end-to-end measure

# Accelerated Graph Processing: 100X+ Speed and 50X Cost Reduction

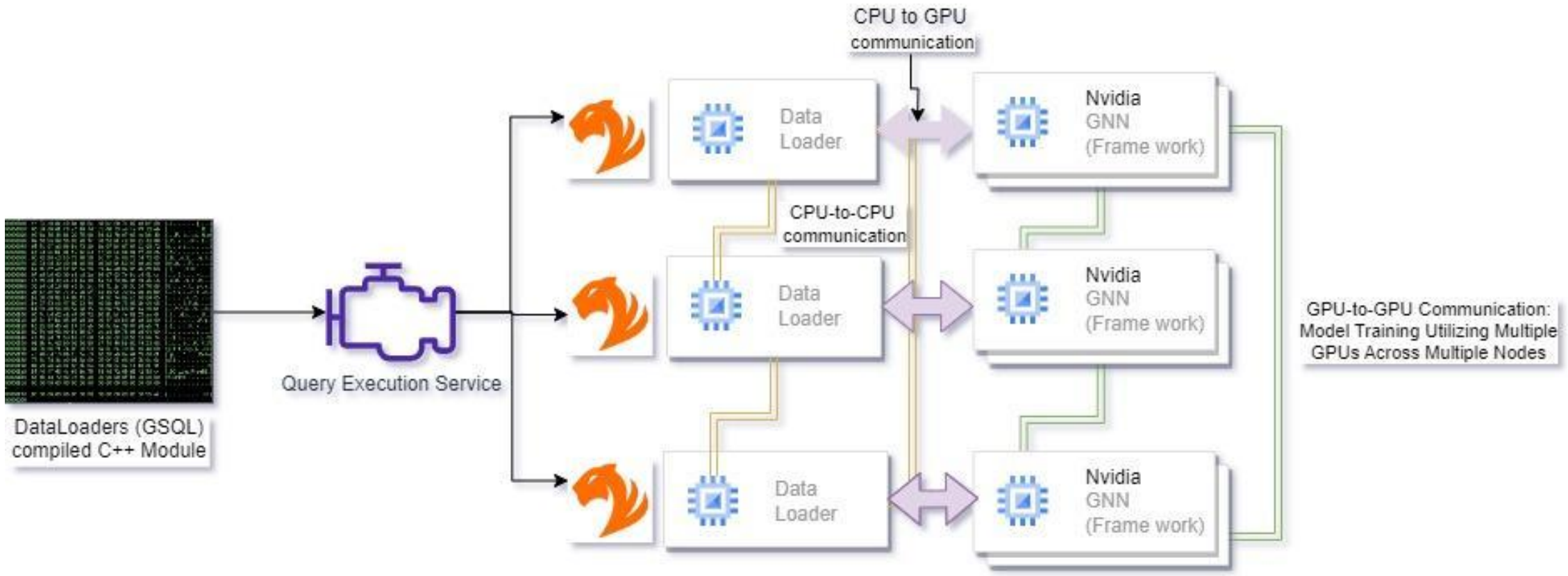
| Instance name | Total Nodes | On-Demand hourly rate($) | vCPU | Memory |
|---|---|---|---|---|
| m7a.32xlarge | 2 | 7.41888 | 128 | 512 GiB |
| p4d.24xlarge | 1 | 32.77 | 96 | 1152 GiB |

| GPU Specifications | |
|---|---|
| GPU – A100 | GPU memory |
| 8 | 320 GB HBM2 |

| Graph | TigerGraph Cluster (Sec) | cuGraph + TigerGraph (Native) (Sec) | CPU Cost | GPU Cost | Benefit(X) |
|---|---|---|---|---|---|
| Graph 22 | 311.162 | 6.91 (45X) | $1.28 | $0.06 | 20 |
| Graph 23 | 617.82 | 9.04 (68X) | $2.55 | $0.08 | 31 |
| Graph 24 | 1205.34 | 14.69 (82X) | $4.97 | $0.13 | 37 |
| Graph 25 | 2888.74 | 21.09 (137X) | $11.91 | $0.19 | 62 |
| Graph 26 | 4842.4 | 41.01 (118X) | $19.96 | $0.37 | 53 |

HBM2 is “High Bandwidth Memory”

# Scaling Accelerated Graph Neural Networks Using the RAPIDS Framework



# Summary and Call to Action

By combining TigerGraph and NVIDIA GPUs we can save consumers billions of dollars each year

**Visit TigerGraph:**

http://tigergraph.com

**Connect with Dan McCreary on LinkedIn:**

https://www.linkedin.com/in/danmccreary/