TigerGraph

ARCHITECTURE OVERVIEW

Why TigerGraph is 100x faster than competition

July 22, 2020

Today's Presenters

Victor Lee

Head of Product Strategy & Developer Relations

- BS in Electrical Engineering and Computer Science from UC Berkeley
- MS in Electrical Engineering from Stanford University
- PhD in Computer Science from Kent State University focused on graph data mining
- 20+ years in tech industry

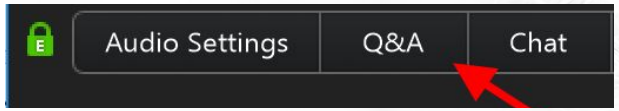
Rayees Pasha

Principal Product Manager

- MS in Computer Science from University of Memphis
- Prior Lead PM and ENG positions at Workday, Hitachi and HP
- Expertise in Database Management and Big Data Technologies

Some Housekeeping Items

- Although your phone is muted you can ask questions at any time using the Q&A tab in the Zoom menu
- The webinar is being recorded and will be emailed you with the slides
- If you have any issues with Zoom please contact the organizer via chat



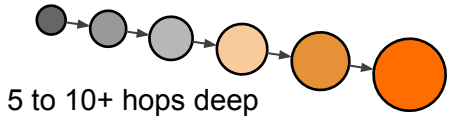
PLEASE SEND QUESTIONS VIA ZOOM Q&A

Today's Outline

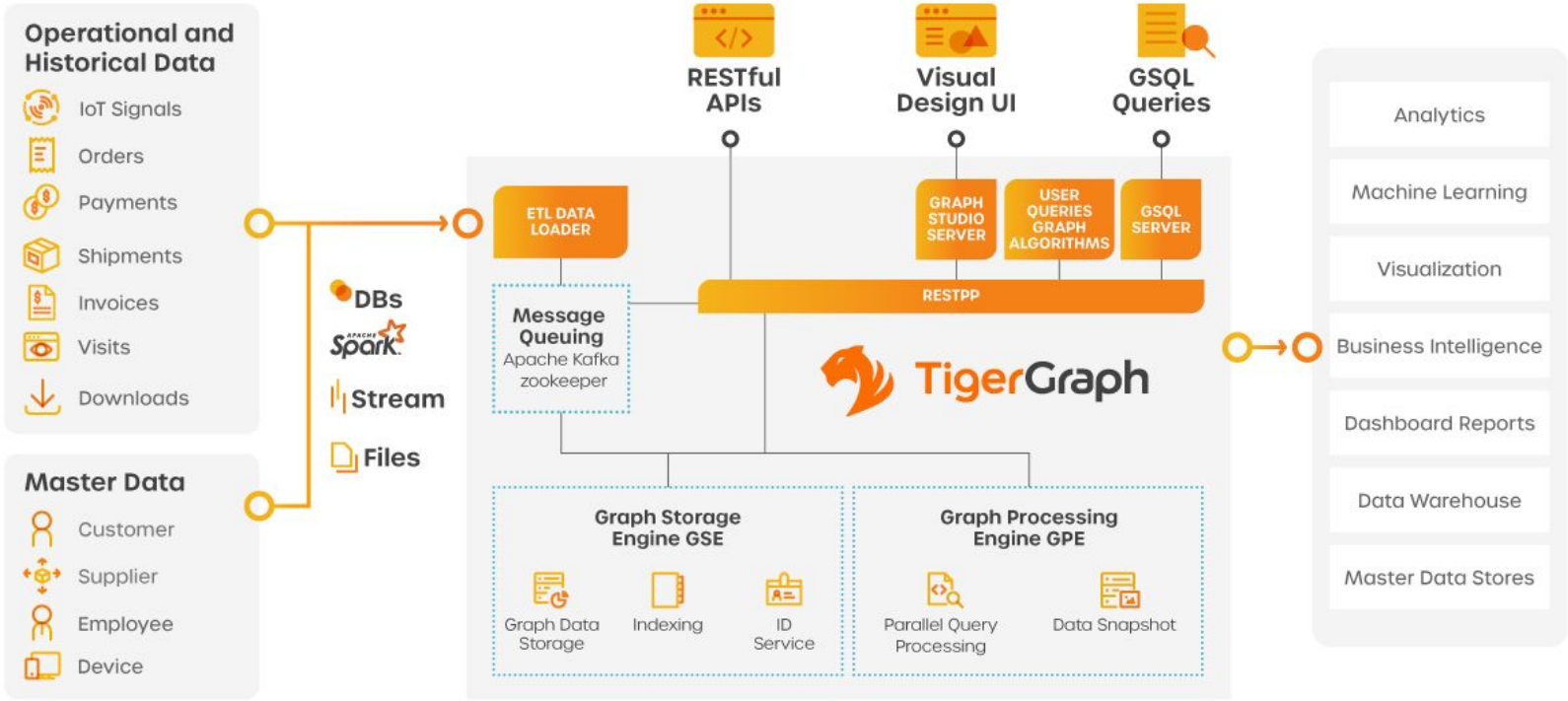
1. System Architecture Overview
2. Data Ingestion and Storage
3. Data Processing
4. Non-functional Features
   - HA
   - Transaction Management
   - Security

SYSTEM ARCHITECTURE OVERVIEW

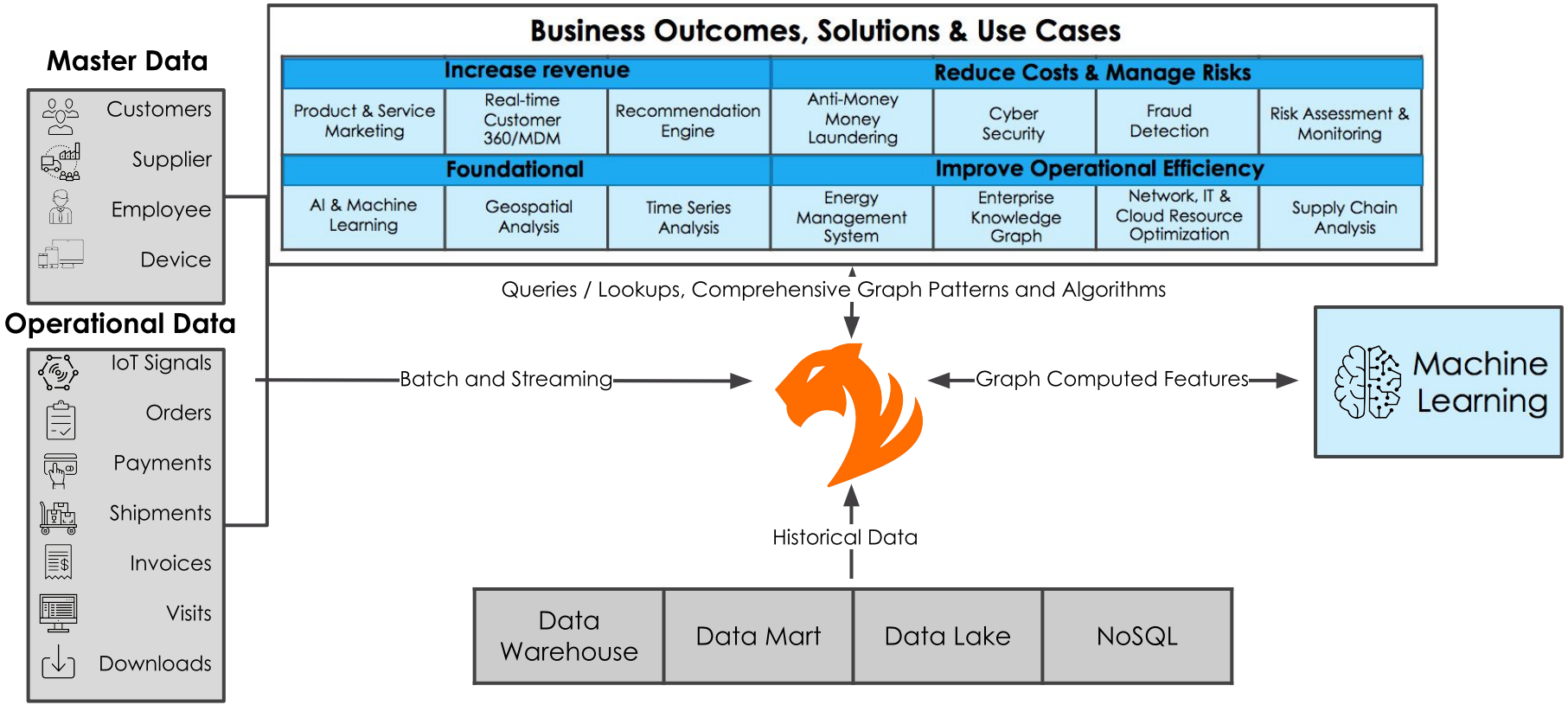
The TigerGraph Difference

Feature	Design Difference	Benefit
Real-Time Deep-Link Querying  	• Native Graph design • C++ engine, for high performance • Storage Architecture	• Uncovers hard-to-find patterns • Operational, real-time • HTAP: Transactions+Analytics
Handling Massive Scale 	• Distributed DB architecture • Massively parallel processing • Compressed storage reduces footprint and messaging	• Integrates all your data • Automatic partitioning • Elastic scaling of resource usage
In-Database Analytics 	• GSQL: High-level yet Turing-complete language • User-extensible graph algorithm library, runs in-DB • ACID (OLTP) and Accumulators (OLAP)	• Avoids transferring data • Richer graph context • In-DB machine learning

TigerGraph Architecture



Seamless integration enables businesses to accomplish more with existing investments

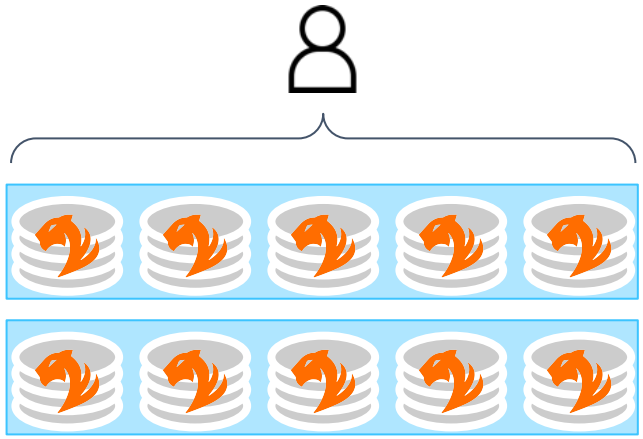


TigerGraph

TigerGraph Distributed Database Architecture

Simple setup, Performant design

- **Setup**: Just tell TigerGraph how many servers.
- TigerGraph seamlessly distributes data.
- Users see a single database, not shards.



Real-time active replication for High Availability (HA)

- write to all
- read from any
- strong consistency

Advantages:

- **Simple to setup and manage**
- **Unlimited scale-out;** simple to expand
- **Scalable OLAP**: massively parallel processing
- **Scalable OLTP**: concurrent ACID transactions
- **Economical**

DATA INGESTION AND STORAGE

Data Ingestion

Step 1

Loaders take in user source data.

- Bulk load of data files or a Kafka stream in CSV or JSON format
- HTTP POSTs via REST services (JSON)
- GSQL Insert commands

Step 2

Dispatcher takes in the data ingestion requests in the form of updates to the database.

1. Query IDS to get internal IDs
2. Convert data to internal format
3. Send data to one or more corresponding GPEs

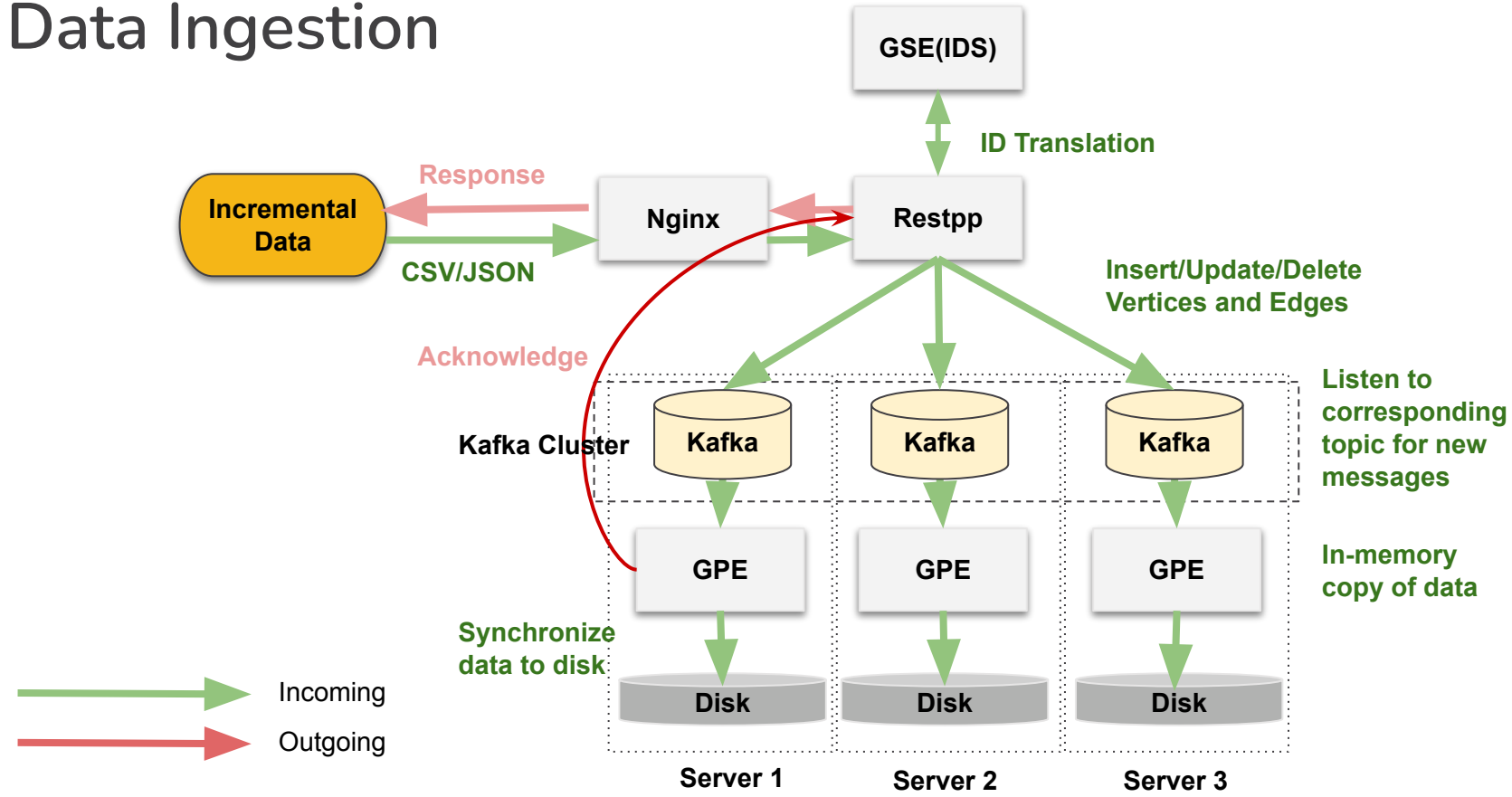
Step 3

Each GPE consumes the partial data updates, processes it and puts it on disk.

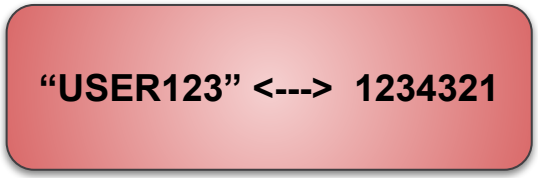
Loading Jobs and POST use UPSERT semantics:

- If vertex/edge doesn't yet exist, create it.
- If vertex/edge already exists, update it.
- Idempotent

Data Ingestion

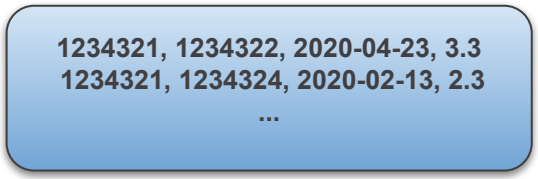


TigerGraph Distributed Native Graph Storage



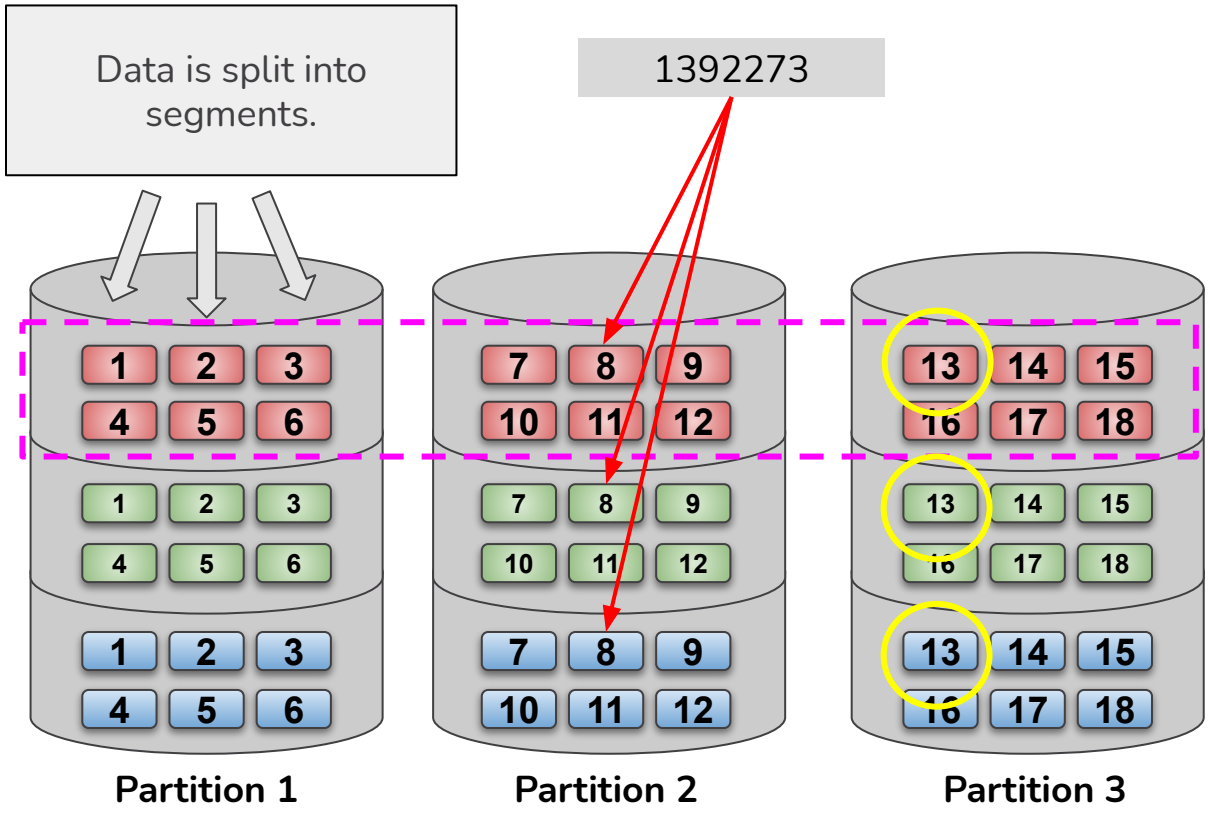
IDS: Bidirectional external ID to Internal ID mapping

Vertex Partitions: Vertex internal ID and attributes



Edge Partitions: Source vertex internal ID, target vertex internal ID, edge attributes

TigerGraph Distributed Native Graph Storage



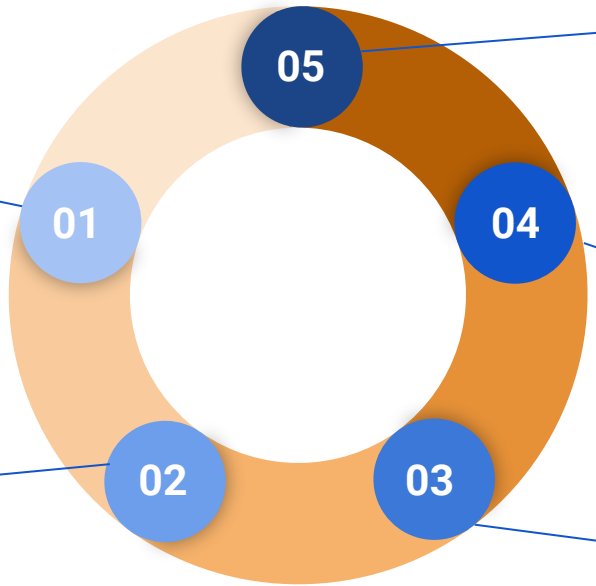
Data in segments is distributed across the cluster.

The segments with same ID store data for the same set of vertices.

The location of a vertex can be calculated based on its internal ID

DATA PROCESSING

Query Processing Workflow Overview



Users use GSQL, GraphStudio or RESTful API to submit a query using HTTP GET/POST request

One of the REST servers parses the request based on graph schema and forwards it to a dispatcher

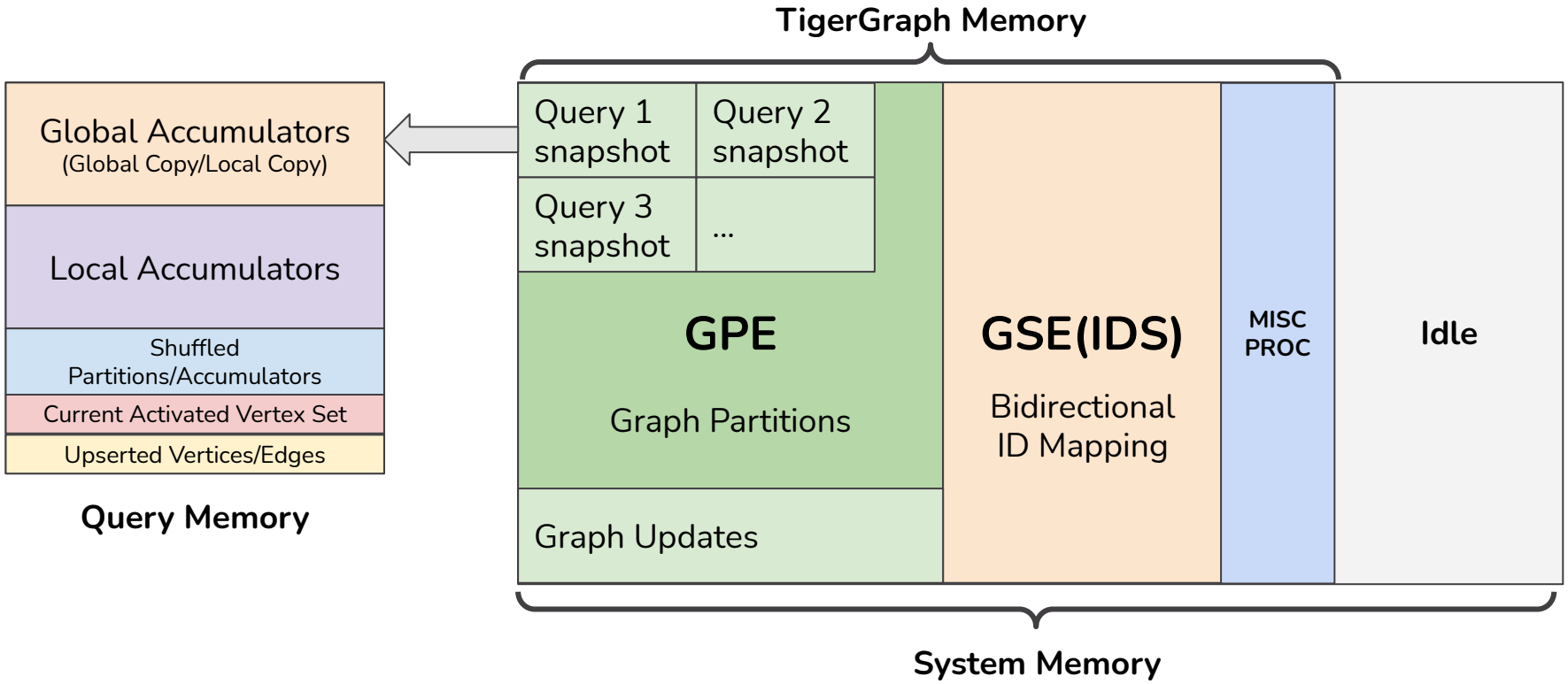
REST server responds back the HTTP request, in JSON format

GPE performs the computation.

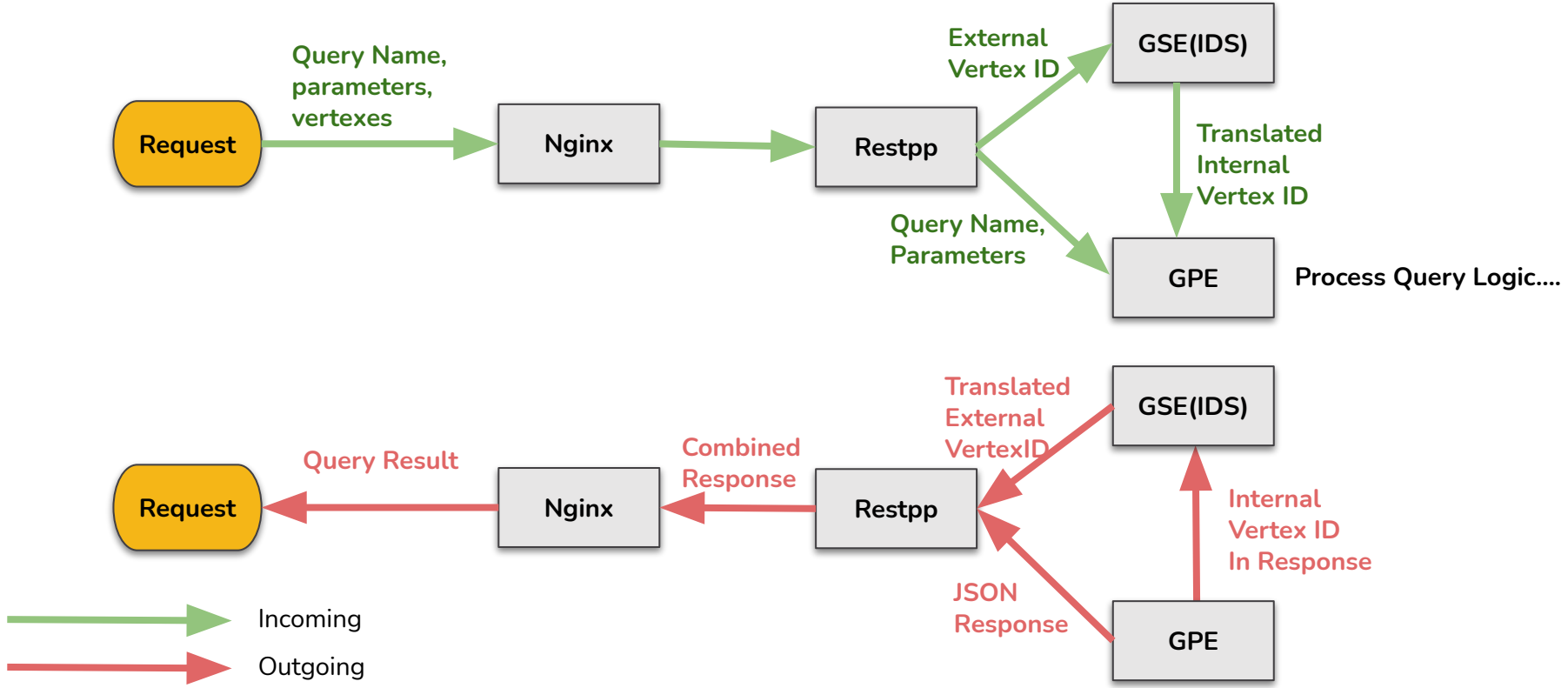
Dispatcher collects and returns the final GPE run result to REST server

Dispatcher queries ID Service, convert the request into internal format, and route it to a GPE

TigerGraph Memory Usage Overview



Query Processing workflow



GSQL Queries

Schema-Based

Optimizes storage efficiency and query speed. Supports data-independent app/query development.

Built-in High Performance Parallelism

Achieves fast results while being easy to code. Accumulators turbocharge parallel computation.

SQL-Like

Familiar to 1 million users

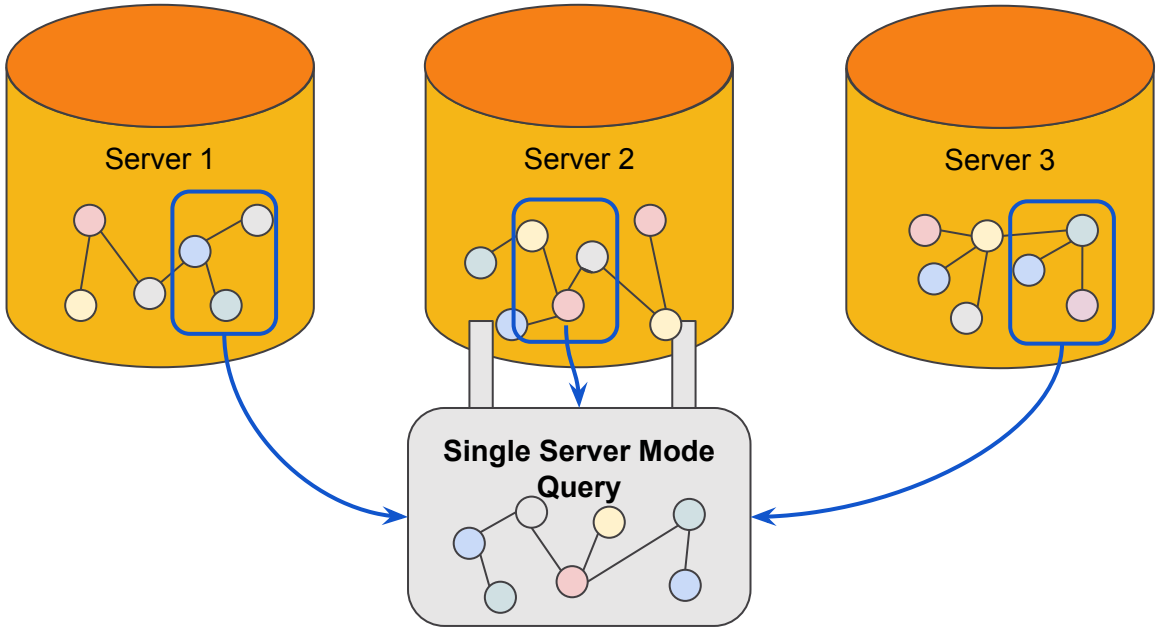
Conventional Control Flow (FOR, WHILE, IF/ELSE)
Makes it easy to implement conventional algorithms

Procedural Queries

Parameterized queries are flexible and can be used to build more complex queries

Transactional Graph Updates
HTAP - Hybrid Transactional / Analytical Processing with real-time data updates

MPP - Distributed Cluster in Single Server mode



Single Server Mode

- The cluster elects one server to be the master for that query.
- All query computations take place on query master.
- Vertex and edge data are copied to the query master as needed.
- Best for queries with one or a few starting vertices.

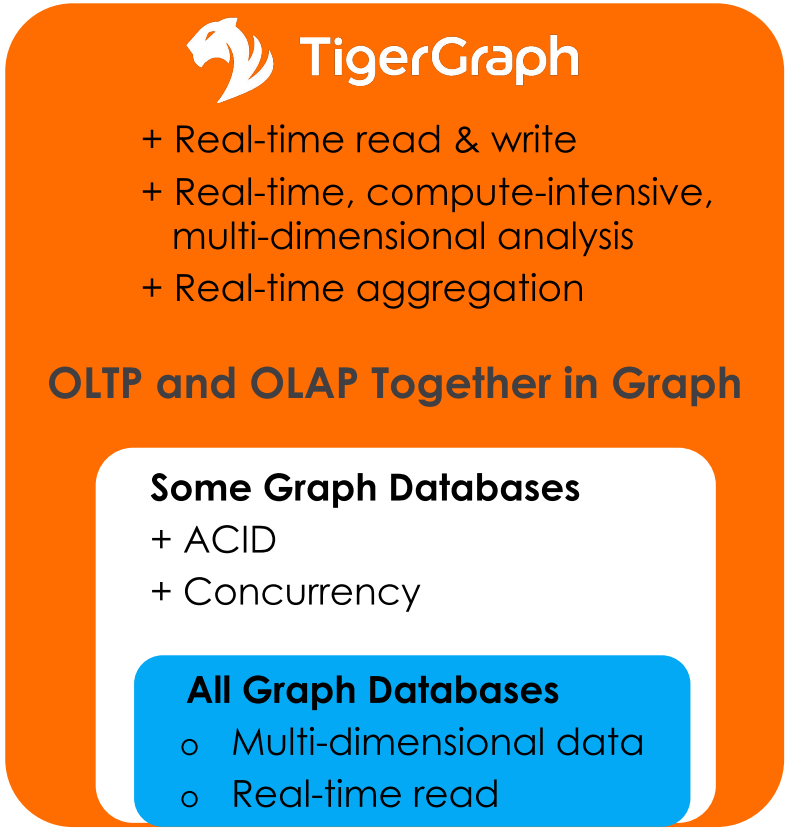
HTAP: Hybrid Transaction/Analytical Processing

OLTP - Transactional

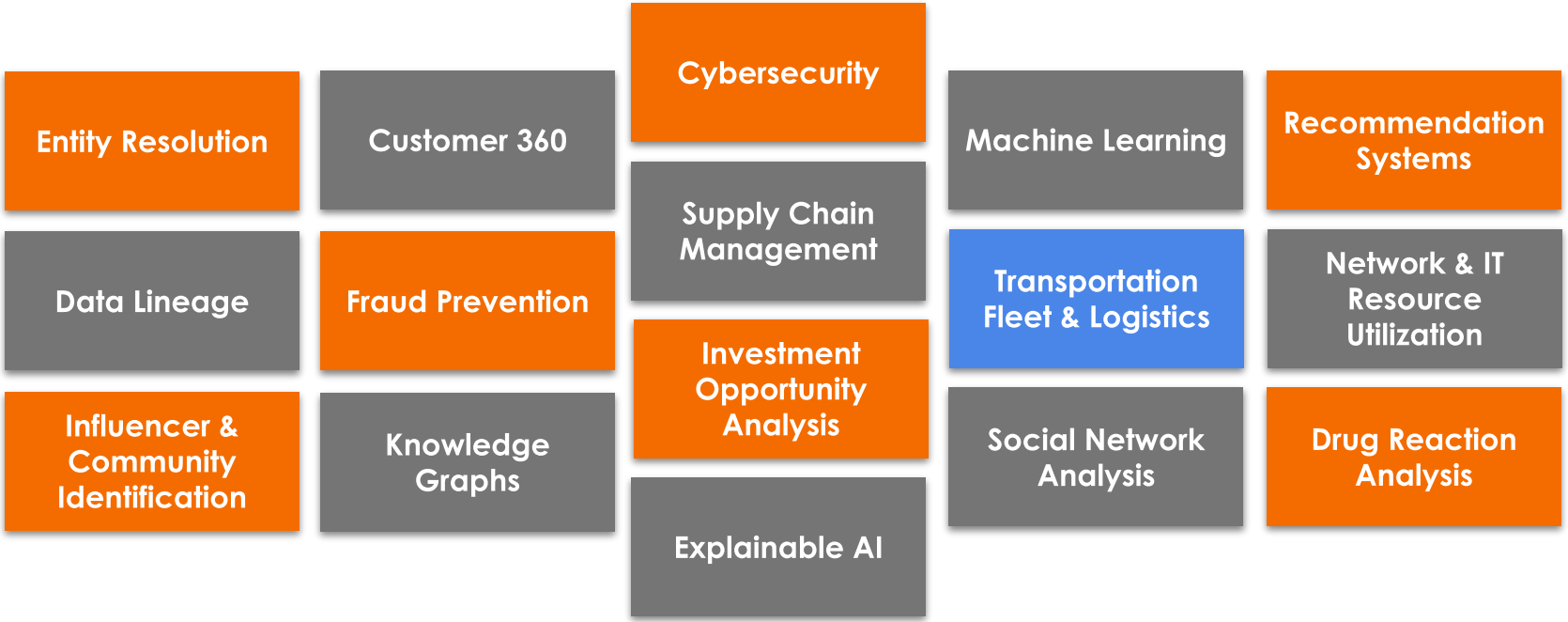
- Real-time read and write
- ACID properties
  (guarantee that transaction is correct)
- Concurrency
  (many transactions at the same time)

OLAP - Analytical

- Multi-dimensional Analysis
- Compute-intensive
- Data-intensive
- Aggregation

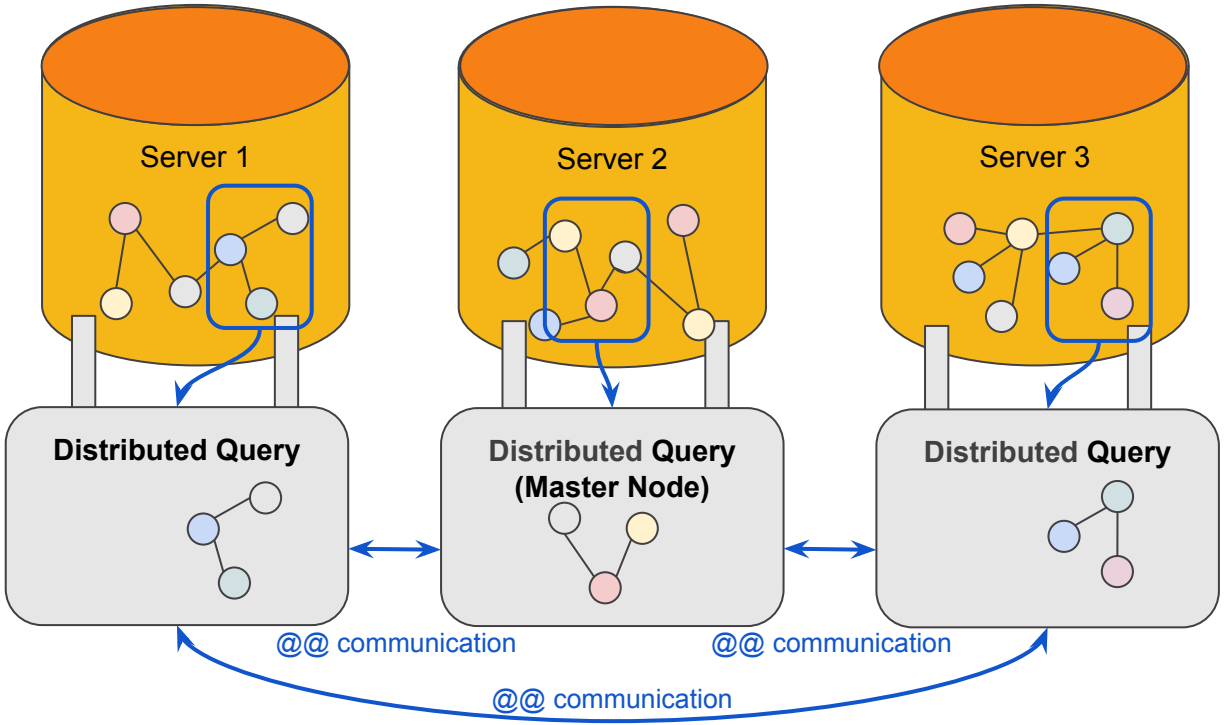


There's No Limit To Where Graph Can Help



TigerGraph

MPP - Distributed Cluster in Distributed mode

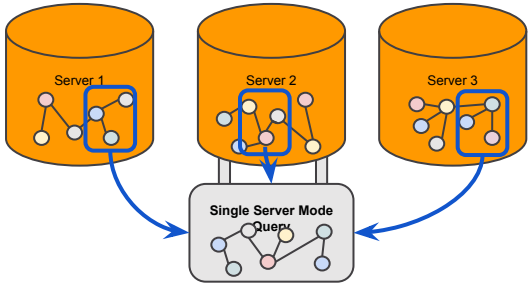


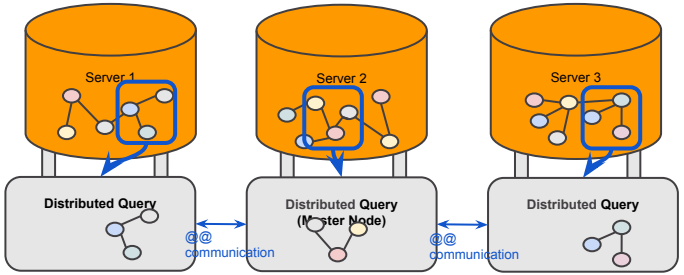
Distributed Mode

- The server that receives the query becomes the master.
- Computations execute on **all** servers in parallel.
- Global accumulators are transferred across the cluster.
- If your query starts from all or most vertices, use this mode.

MPP mechanism

Single Server mode VERSUS Distributed mode





Single Server Mode is better when

1. Starting from a single or small number of vertices.
2. Modest number of vertices/edges are traversed.
3. Heavy usage of global accumulators.

Ex: Point query, single entity-based transaction/update

Distributed Mode is better when

1. Starting from all or a large number of vertices.
2. Very large number vertices/edges are traversed.

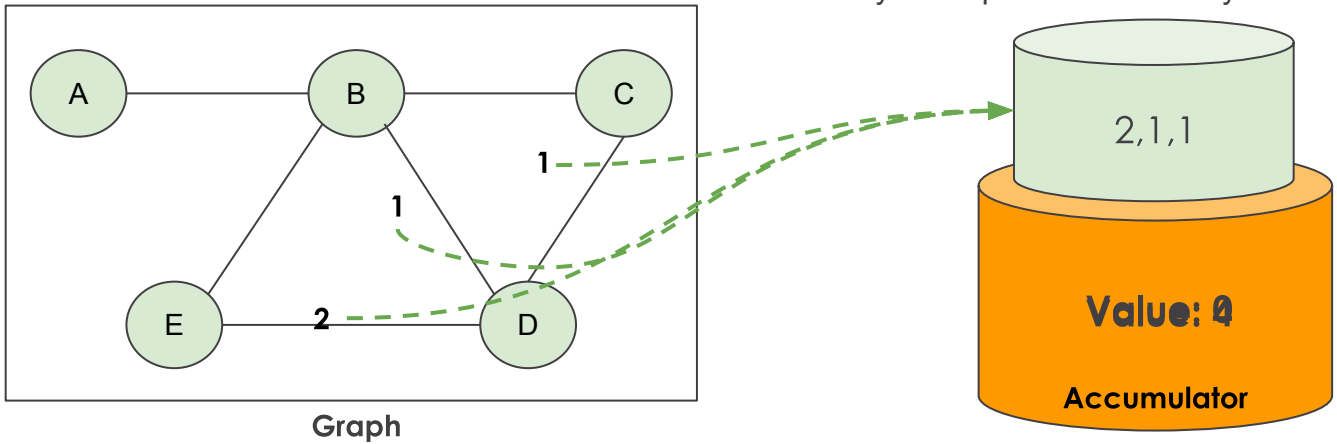
Ex: Most graph algorithms & global analytics (PageRank, Closeness Centrality, Louvain Community, etc.)

Accumulators

Accumulators are a special type of variables that accumulate information from multiple workers/threads during a query. Workers/threads work asynchronously.

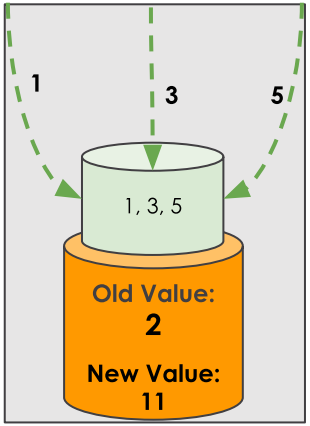
Accumulating phase 1: Receive messages and store them temporarily in a bucket that belongs to the accumulator.

Accumulating phase 2: Aggregate the messages it received based on its accumulator type. The aggregated value will become the accumulator's value which can be accessed by other parts of the Query.



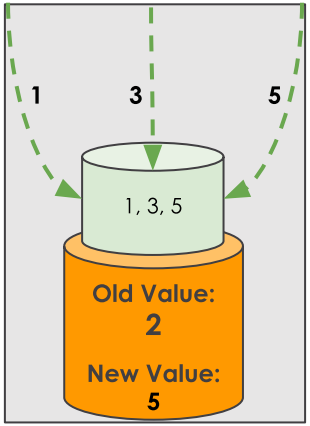
Accumulators

The GSQL language provides many different accumulators, which follow the same rules for receiving and accessing data. However, each of them has their unique way of **aggregating values.**



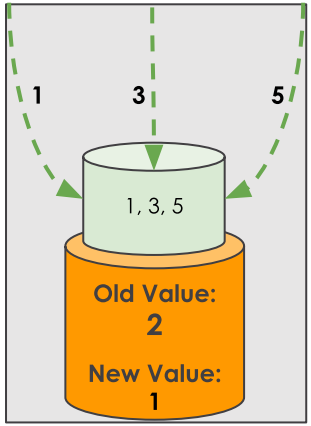
SumAccum<int>

Computes and stores the cumulative sum of numeric values or the cumulative concatenation of text values.



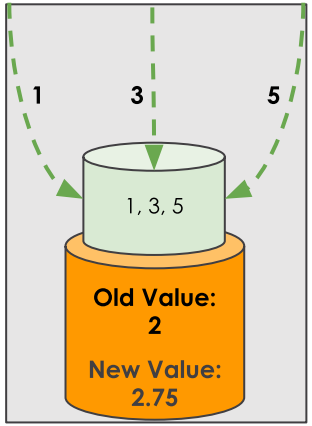
MaxAccum<int>

Computes and stores the cumulative maximum of a series of values.



MinAccum<int>

Computes and stores the cumulative minimum of a series of values.



AvgAccum

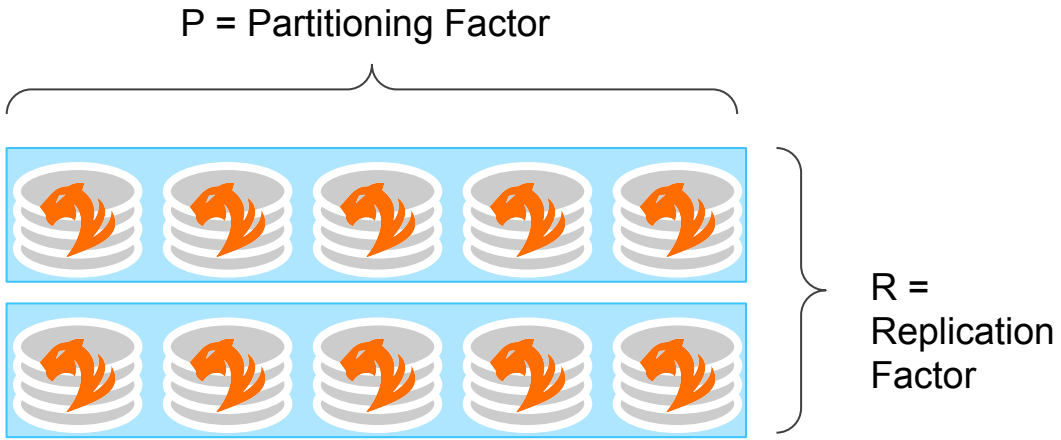
Computes and stores the cumulative mean of a series of numeric values.

NON-FUNCTIONAL FEATURES

- High Availability
- Access Control & Security
- Transaction Management

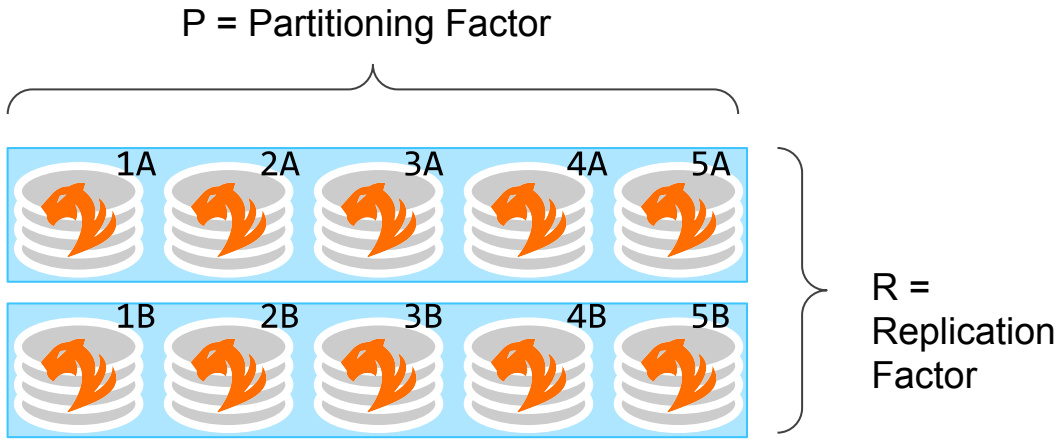
High Availability

- TigerGraph HA Replication provides both Increased Throughput and Continuous Operation
- Cluster size = P X R (Partitions x Replicas)
- Any cluster size is allowed, except 1x2



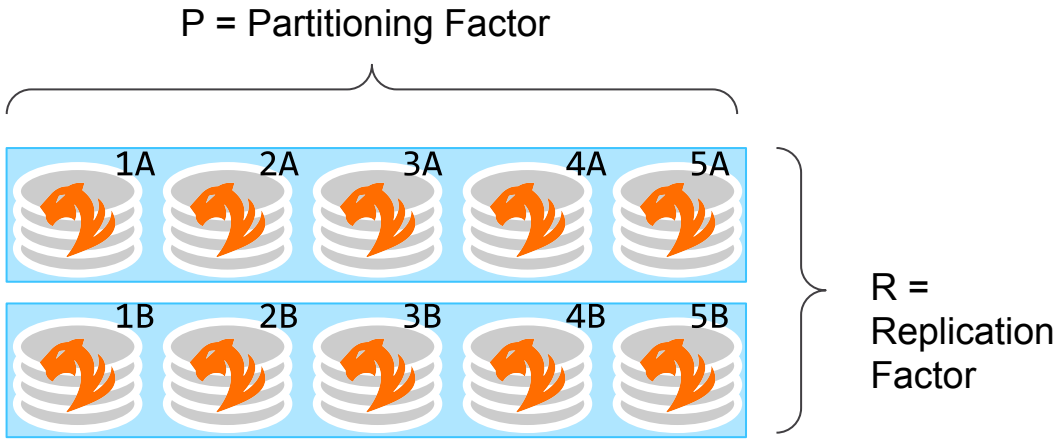
HA and Concurrency

- Each server has T available workers for serving requests (GSQL query, REST POST, etc.)
  T is a system configuration parameter, defaults to 8. Consider number of CPU cores.
- Cluster's total number of workers = TxPxR, e.g. 8x5x2 = 80
  - A point mode query uses 1 worker.
  - A distributed mode query use P workers.



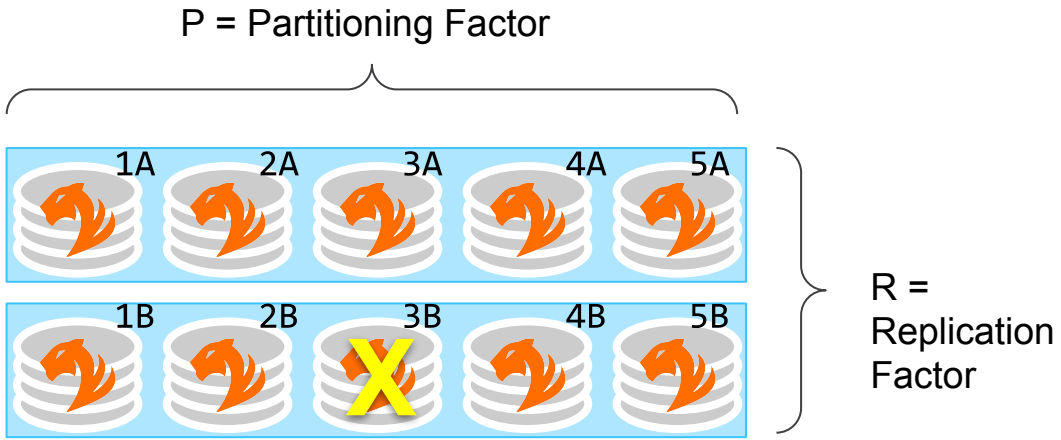
HA Read and Write Behavior

- All Replicas are Read/Write, always in sync with the latest updates
- Writes go to all replicas (e.g. both 1A and 1B).
- Reads can be from any one replica (e.g. either 1A or 1B).
- Distributed queries can mix replicas (e.g. {1A, 2B, 3B, 4A, 5B} is a valid active set for a request.)



HA Continuous Operation

- If any single server is unavailable (expected or unexpected):
  - When it fails to respond after a certain number of tries, requests will automatically divert to another replica (e.g. 3B is unavailable, so use 3A)
  - If it fails in the middle of a transaction, that transaction might be aborted.
- System continues to operation, with reduced throughput, until server is restored.



NON-FUNCTIONAL FEATURES

- High Availability
- Access Control & Security
- Transaction Management

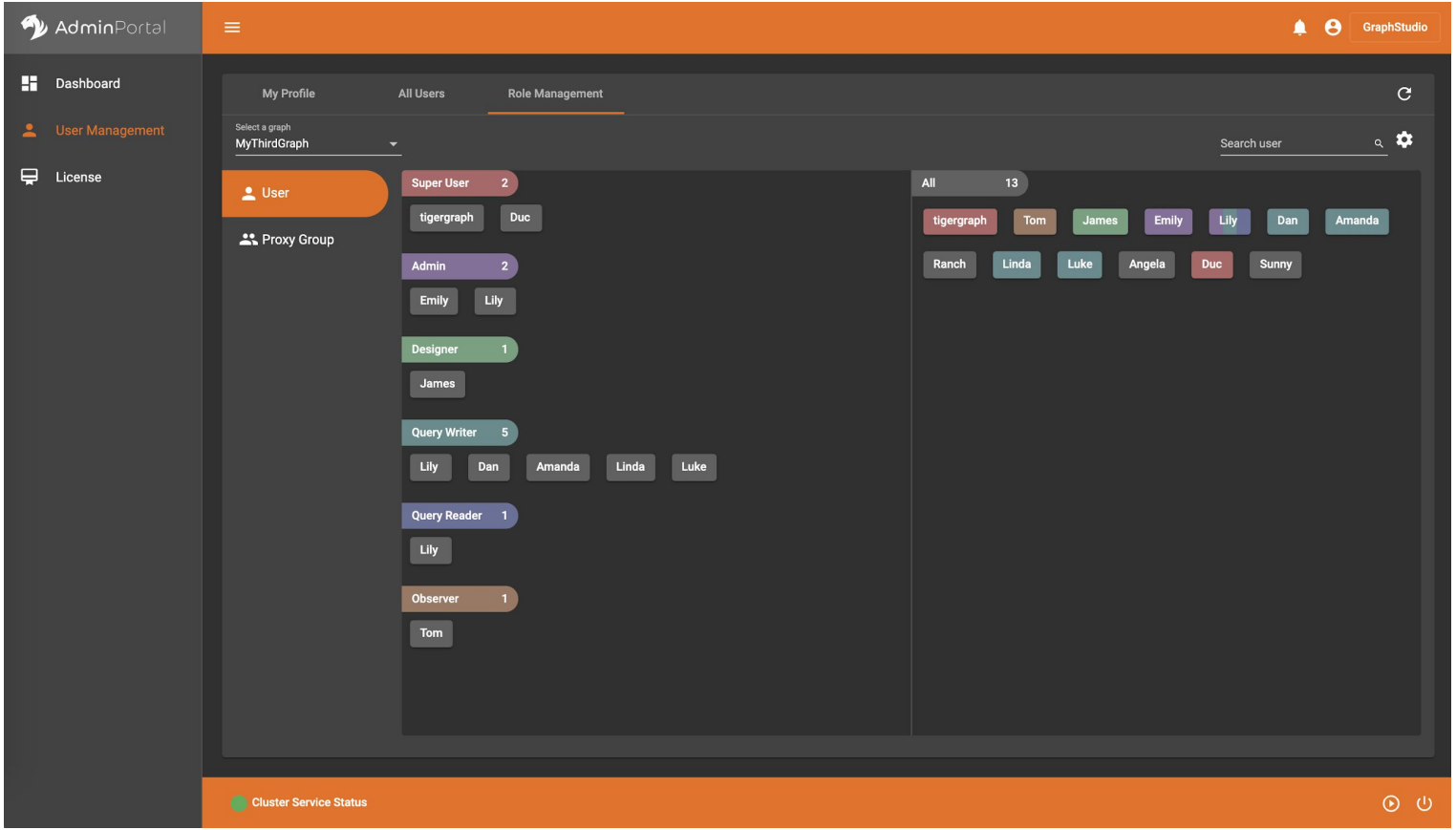
Role-Based Access Control

- Follows SQL approach for roles.
  - GSQL:

```
GRANT <role> ON GRAPH <graph> TO <user1, user2, ... >
REVOKE <role> ON GRAPH <graph> FROM <user1, user2, ... >
```

- Can map TigerGraph roles to external LDAP roles and groups.

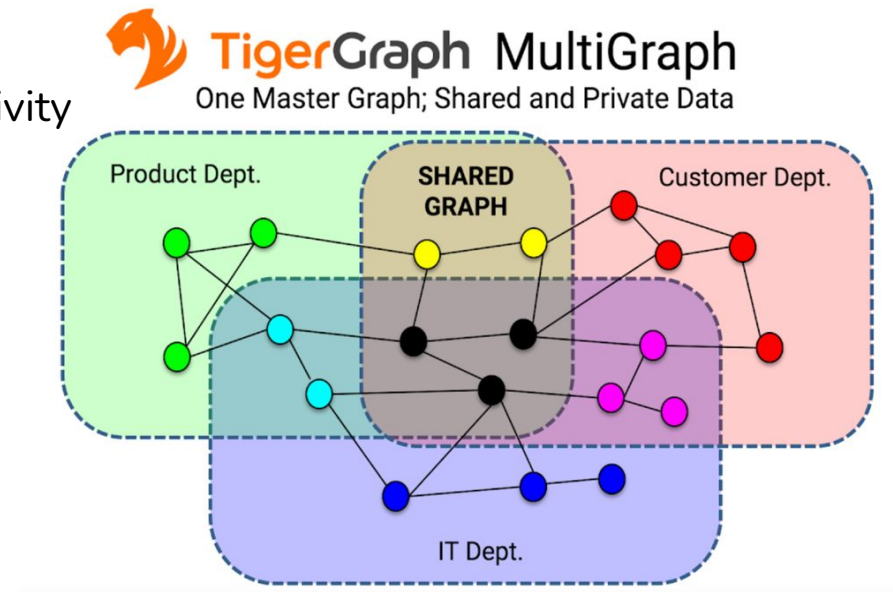
Admin Portal UI for Managing User Privileges



TigerGraph

MultiGraph for RBAC and Data Sharing

- **Share & Collaborate**
  - Multiple groups share one master database
    ⇒ data integration, insights, productivity
- **Real-time, Updatable**
  Shared updates, no copying
  ⇒ cleaner, faster, cheaper, safer
- **Fine-Grained Security**
  - Each group is granted its own view
  - Each group has its own admin user, who manages local users' privileges.



Roles and Privileges

Built-In Roles:

- **Queryreader:** run existing loading jobs & queries for its assigned graph.
- **Querywriter:** Queryreader privileges + create queries and run data-manipulation commands on its assigned graph.
- **Designer:** Querywriter privileges + modify the schema, create loading jobs for its assigned graph.
- **Globaldesigner:** Designer privileges + create global schema, create objects. Also, delete graphs which they created.
- **Admin:** Designer privileges, + create/drop users, grant/revoke roles for its assigned graph. That is, control existence & privileges of its local users.
- **Superuser:** admin privileges on all graphs. Create global vertex & edge types, create multiple graphs, and clear the database.

User-Defined Roles: in development

Command Type	Operations	super-user	admin	global-designer	designer	query-writer	query-reader
Status	Ls	x	x	x	x	x	x
User Management	Create/Drop User	x	x	-	-	-	-
	Show User	x	x	x	x	x	x
	Alter (Change) Password	x	x	x	x	x	x
	Grant/Revoke Role	x	x	-	-	-	-
	Create/Drop/Show Secret	x	x	x	x	x	x
	Create/Drop/Show/ Refresh Token (Deprecated)	x	x	x	x	x	x
Schema Design	Create/Drop Vertex/Edge/Graph	x	-	x	-	-	-
	Clear Graph Store	x	-	-	-	-	-
	Drop All	x	-	-	-	-	-
	Use Graph	x	x	x	x	x	x
	Use Global	x	x	x	x	x	x
	Create/Run Global Schema_Change Job	x	-	x	-	-	-
	Create/Run Schema_Change Job	x	x	x	x	-	-
Loading and Querying	Create/Drop Loading Job	x	x	x	x	-	-

Data Encryption

- Encrypted Data at Rest
  - Choice of encryption levels (file, volume, partition, disk)
    - Kernel level: dm-crypt / cryptsetup
    - User level: FUSE (Filesystem in User Space)
  - Automatically encrypted in TigerGraph Cloud

- Encrypted Data in Transit
  - Can set up SSL/TLS for HTTPS protocol
  - Automatically encrypted in TigerGraph Cloud

NON-FUNCTIONAL FEATURES

- High Availability
- Access Control & Security
- Transaction Management

Transactional Model

- The TigerGraph distributed database provides full ACID transactions with sequential consistency

- Transactions definition:
  - Each GSQL Query procedure is a transaction. Each query may have multiple SELECT, INSERT, or UPDATE statements.

  - Each REST++ GET, POST, or DELETE operation (which may have multiple update operations within it) is a transaction.

ACID Compliance

Atomicity	Consistency	Isolation Level	Durability
GSQL query w/ or w/o updates = Transaction Transactions are “all or nothing”: either all changes are successful, or none is successful.	Single-server Consistency: A transaction obeys data validation rules. Distributed System Sequential Consistency: Every replica of the data performs the same operations in the same order.	Repeatable Read: • Each transaction sees the same data. No Dirty/Phantom Read: • A transaction's updates are not visible to other transactions until the update is committed.	The TigerGraph platform implements write-ahead logging (WAL) to disk to provide durability. Logs are consumed periodically to update the database on disk.

Resources

TigerGraph Platform Overview:
https://docs.tigergraph.com/intro/tigergraph-platform-overview

HA Cluster Configuration:
https://docs.tigergraph.com/admin/admin-guide/installation-and-configuration/ha-cluster

Transaction Processing and ACID Support:

https://docs.tigergraph.com/dev/transaction-and-acid

MultiGraph:

https://docs.tigergraph.com/intro/multigraph-overview

TigerGraph

Get Started for Free

- Try TigerGraph Cloud
- Download TigerGraph's Developer Edition
- Take a Test Drive - Online Demo
- Get TigerGraph Certified
- Join the Community

@TigerGraphDB /tigergraph /TigerGraphDB /company/TigerGraph

BACKUP