TigerGraph

TIGERGRAPH와 NEO4J를 바라보는 Architect 관점 그리고 산업계 동향

윤명식 메가존클라우드

MEGAZONE CLOUD

Who am I

윤명식 매니저

- MegazoneCloud OCTO
- Graph Database Architect
- TigerGraph DB Specialist
- 01089489592
- myoungsig.youn@mz.co.kr
- jazzlian@gmail.com

산업계 동향

Graph DB Ranking

Rank			DBMS	Database Model	Score		
Sep 2023	Aug 2023	Sep 2022			Sep 2023	Aug 2023	Sep 2022
1.	1.	1.	Neo4j	Graph	50.39	-1.03	-9.09
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	35.45	+0.45	-5.22
3.	3.	4.	Virtuoso	Multi-model	5.38	+0.59	-0.57
4.	4.	5.	OrientDB	Multi-model	4.33	-0.36	-0.48
5.	5.	3.	ArangoDB	Multi-model	4.29	-0.08	-1.74
6.	6.	19.	Memgraph	Graph	2.88	0.00	+2.48
7.	7.	8.	GraphDB	Multi-model	2.60	-0.10	+0.07
8.	8.	6.	Amazon Neptune	Multi-model	2.54	-0.13	-0.66
9.	9.	7.	JanusGraph	Graph	2.39	-0.01	-0.25
10.	11.	14.	NebulaGraph	Graph	2.33	-0.01	+1.16
11.	10.	10.	Stardog	Multi-model	2.28	-0.08	+0.61
12.	12.	9.	TigerGraph	Graph	2.21	-0.10	+0.07
13.	15.	12.	Dgraph	Graph	1.89	+0.19	+0.41
14.	13.	11.	Fauna	Multi-model	1.69	-0.10	+0.13
15.	14.	13.	Giraph	Graph	1.65	-0.11	+0.45
16.	16.	15.	AllegroGraph	Multi-model	1.15	-0.10	+0.02
17.	17.	17.	Blazegraph	Multi-model	1.02	-0.17	+0.13
18.	18.	18.	TypeDB	Multi-model	1.02	+0.02	+0.14
19.	20.		SurrealDB	Multi-model	0.87	+0.12	
20.	19.	16.	Graph Engine	Multi-model	0.78	+0.02	-0.15

출처: https://db-engines.com/en/ranking/graph+dbms

Graph Database Infograph

OPERATIONAL GRAPH DATABASES

DBMS products suitable for a broad range of enterprise-level transactional applications.

TITAN

JanusGraph

OrientDB

neo4j

MULTI-MODAL GRAPHS

Encompasses databases designed to support different model types.

Azure Cosmos DB

ArangoDB

sqrrl

REAL-TIME BIG GRAPHS

Enables real-time large graph analysis with both 100M+ vertex or edge traversals/sec/server and 100K+ updates/sec/server.

TigerGraph

KNOWLEDGE GRAPH / RDF

Provides a general method for modeling of syntactic and inference information.

Franz Inc. AllegroGraph

OPENLINK VIRTUOSO

blazegraph

Stardog

ANALYTIC GRAPHS

Focused on solving complex analytical problems, but not in real time.

APACHE GIRAPH

GraphX

RDBMS + GRAPH

ORACLE GRAPH

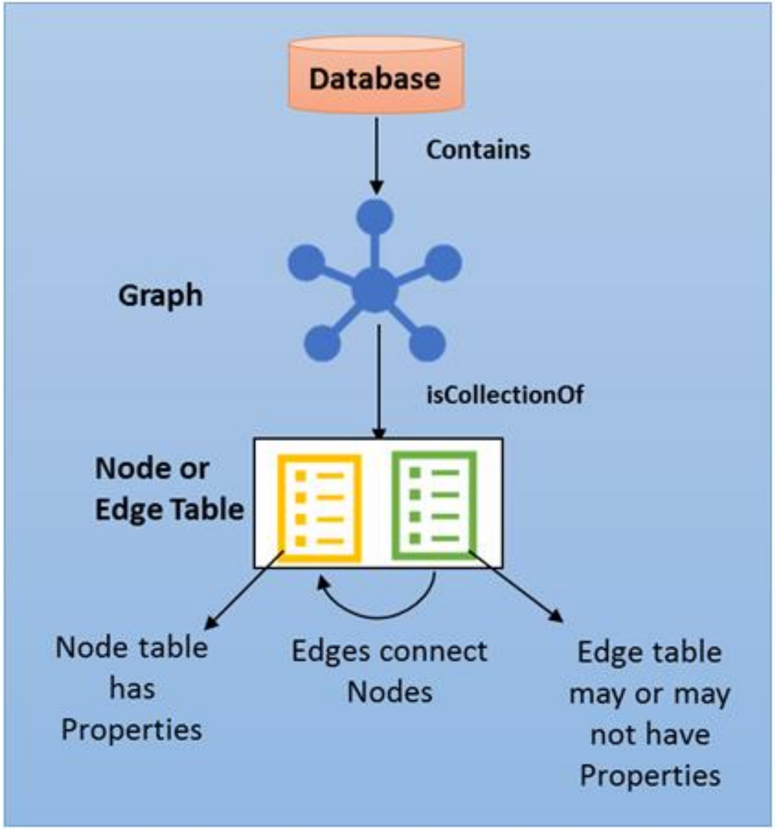
```
CREATE PROPERTY GRAPH BANK_GRAPH
    VERTEX TABLES (
        BANK_ACCOUNTS
        KEY (ID)
        PROPERTIES (ID, Name, Balance)
    )
    EDGE TABLES (
        BANK_TRANSFERS
        KEY (TXN_ID)
        SOURCE KEY (src_acct_id) REFERENCES BANK_ACCOUNTS(ID)
        DESTINATION KEY (dst_acct_id) REFERENCES BANK_ACCOUNTS(ID)
        PROPERTIES (src_acct_id, dst_acct_id, amount)
    );
```

```
REM Check if there are any 3-hop (triangles) transfers that
start and end at the same account
SELECT acct_id, COUNT(1) AS Num_Triangles
    FROM graph_table (BANK_GRAPH
        MATCH (src) - []->{3} (src)
        COLUMNS (src.id AS acct_id)
    ) GROUP BY acct_id ORDER BY Num_Triangles DESC;

   ACCT_ID    NUM_TRIANGLES
__________ ________________
       918                3
       751                3
       534                3
       359                3
       119                2
       677                2
       218                2
…
118 rows selected.
```

Get started with property graphs in Oracle Database 23c Free – Developer Release

Oracle spatial and Graph

SQL Server GRAPH




```sql
-- Create a GraphDemo database
IF NOT EXISTS (SELECT * FROM sys.databases WHERE NAME = 'graphdemo')
    CREATE DATABASE GraphDemo;
GO

USE GraphDemo;
GO

-- Create NODE tables
CREATE TABLE Person (
  ID INTEGER PRIMARY KEY,
  name VARCHAR(100)
) AS NODE;

CREATE TABLE Restaurant (
  ID INTEGER NOT NULL,
  name VARCHAR(100),
  city VARCHAR(100)
) AS NODE;

CREATE TABLE City (
  ID INTEGER PRIMARY KEY,
  name VARCHAR(100),
  stateName VARCHAR(100)
) AS NODE;

-- Create EDGE tables.
CREATE TABLE likes (rating INTEGER) AS EDGE;
CREATE TABLE friendOf AS EDGE;
CREATE TABLE livesIn AS EDGE;
CREATE TABLE locatedIn AS EDGE;
```

출처: https://learn.microsoft.com/ko-kr/sql/relational-databases/graphs/sql-graph-architecture?view=sql-server-ver16

SQL Server GRAPH

```
-- Find Restaurants that John likes
SELECT Restaurant.name
FROM Person, likes, Restaurant
WHERE MATCH (Person-(likes)->Restaurant)
AND Person.name = 'John';

-- Find Restaurants that John's friends like
SELECT Restaurant.name
FROM Person person1, Person person2, likes, friendOf, Restaurant
WHERE MATCH(person1-(friendOf)->person2-(likes)->Restaurant)
AND person1.name='John';

-- Find people who like a restaurant in the same city they live in
SELECT Person.name
FROM Person, likes, Restaurant, livesIn, City, locatedIn
WHERE MATCH (Person-(likes)->Restaurant-(locatedIn)->City AND Person-(livesIn)->City);
```

```
-- Find friends-of-friends-of-friends, excluding those cases where the relationship "loops back".
-- For example, Alice is a friend of John; John is a friend of Mary; and Mary in turn is a friend of Alice.
-- This causes a "loop" back to Alice. In many cases, it is necessary to explicitly check for such loops and exclude the results .
SELECT CONCAT(Person.name, '->', Person2.name, '->', Person3.name, '->', Person4.name)
FROM Person, friendOf, Person as Person2, friendOf as friendOffriend, Person as Person3, friendOf as friendOffriendOfFriend, Person as
Person4
WHERE MATCH (Person-(friendOf)->Person2-(friendOffriend)->Person3-(friendOffriendOfFriend)->Person4)
AND Person2.name != Person.name
AND Person3.name != Person2.name
AND Person4.name != Person3.name
AND Person.name != Person4.name;
```

PostgreSQL + Apache AGE(BITNINE)

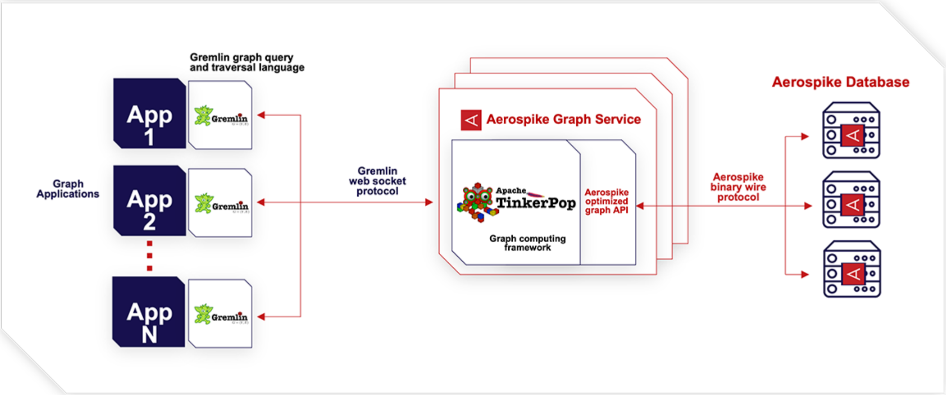
- **Graph Database Plugin for PostgreSQL**
- **Hybrid Queries (OpenCypher And SQL)**
- **Fast Graph Query Processing**
- **Graph Visualization and Analytics**
- **Current PG13 support**

https://age.apache.org/

```
CREATE EXTENSION age;

LOAD 'age’;

SET search_path = ag_catalog, "$user", public;

SELECT create_graph('graph_name’);

SELECT *
FROM cypher('graph_name', $$
    CREATE (:label {property:value})
$$) as (v agtype);

SELECT *
FROM cypher('graph_name', $$
    MATCH (v)
    RETURN v
$$) as (v agtype);

SELECT *
FROM cypher('graph_name', $$
    MATCH (a:Person), (b:Person)
    WHERE a.name = 'Node A' AND b.name = 'Node B'
    CREATE (a)-[e:RELTYPE {name:a.name + '<->' + b.name}]->(b)
    RETURN e
$$) as (e agtype);
```

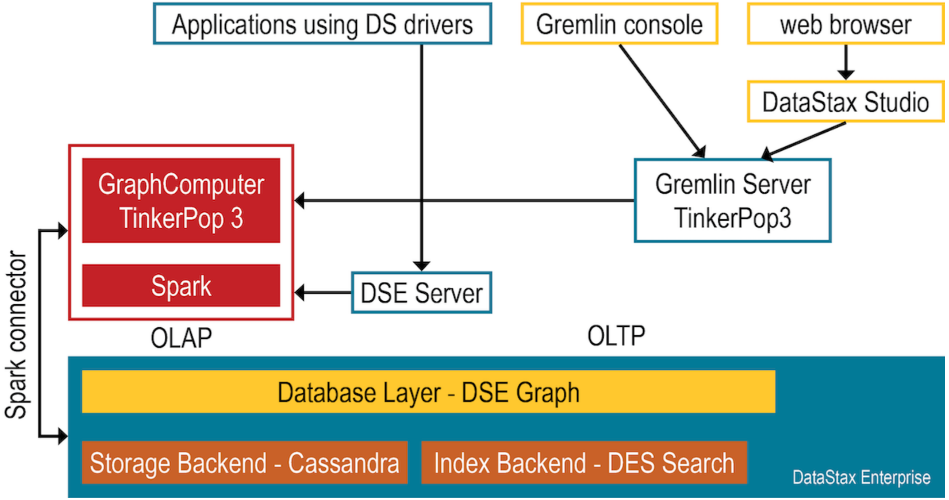
NOSQL + GRAPH

Apache TinkerPop GRAPH



AEROSPIKE GRAPH

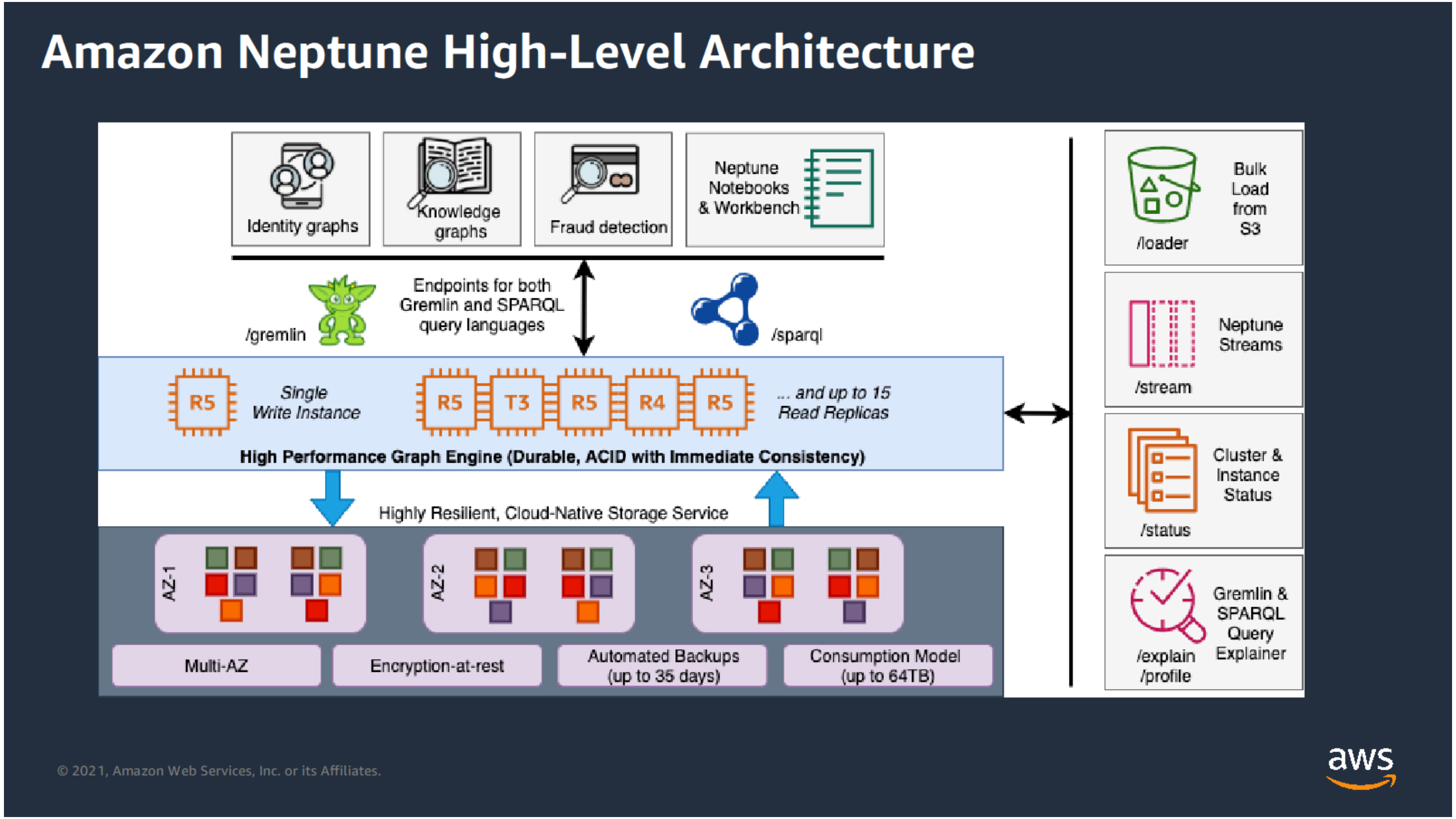
PayPal: Graph on Areospike



DES GRAPH

실무자를 위한 그래프 데이터 활용법

Amazon Neptune



출처: https://aws.amazon.com/ko/blogs/industries/graphing-the-grid-on-aws/

Neo4J Deep Dive

Who is Neo4J?

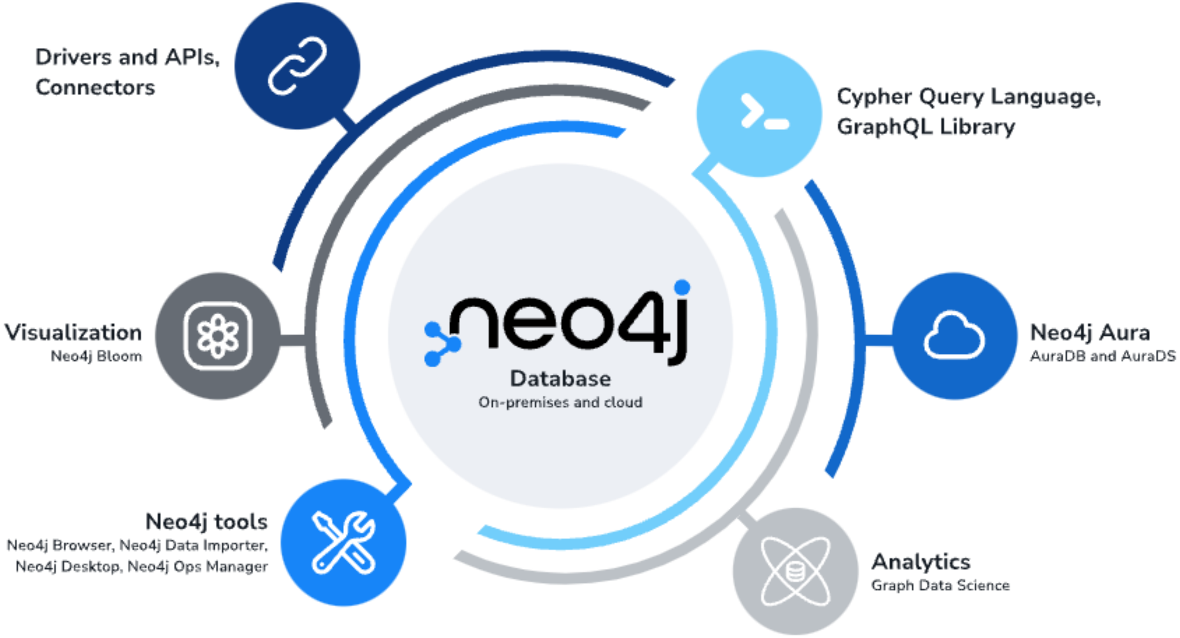


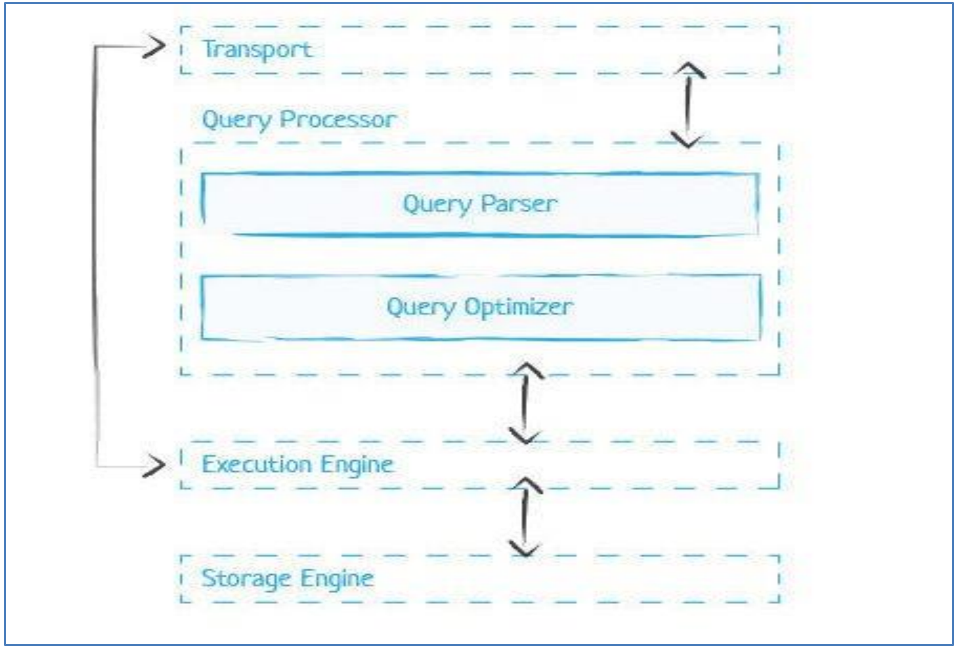
Figure 1. Overview of the Neo4j ecosystem

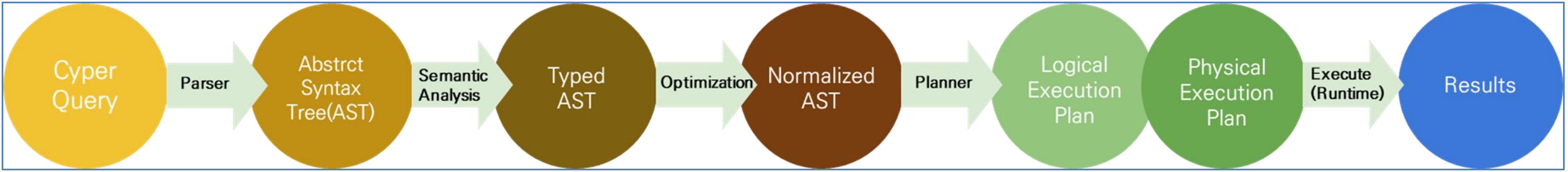
https://www.crunchbase.com/organization/neo-technology

[**Building Knowledge Graphs: A Practitioner's Guide**]()

Query Performance

- Cost-Base Optimizer
- Statistics
- Explain, Profile
- Vector Search
- Second Index
- Full Text Index





출처: https://en-core.com/kor/board/notice?viewMode=view&ca=+tech&sel_search=&txt_search=&page=1&idx=101

Query Performance

Cypher | Copy to Clipboard | Run in Neo4j Browser

```
PROFILE
MATCH (p {name: 'Tom Hanks'})
RETURN p
```

```
+-----------------------------------------+
| p |
+-----------------------------------------+
| (:Person {name: "Tom Hanks", born: 1956}) |
+-----------------------------------------+

+------------------------------------------------------------------------------------------------------+
| Plan      | Statement   | Version       | Planner | Runtime      | Time   | DbHits | Rows | Memory (Bytes) |
+------------------------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.3" | "COST"  | "PIPELINED" | 26     | 406    | 1    | 136            |
+------------------------------------------------------------------------------------------------------+

+------------------------+-------------------------+----------------+------+---------+----------------+------------------------+-----------+---------------------+
| Operator               | Details                 | Estimated Rows | Rows | DB Hits | Memory (Bytes) | Page Cache Hits/Misses | Time (ms) | Other |
+------------------------+-------------------------+----------------+------+---------+----------------+------------------------+-----------+---------------------+
| +ProduceResults@neo4j  | p                       | 8              | 1    | 3       |                |                        |           | Fused in Pipeline 0 |
| |                      +-------------------------+----------------+------+---------+----------------+------------------------+-----------+---------------------+
| +Filter@neo4j          | p.name = $autostring_0 | 8              | 1    | 239     |                |                        |           | Fused in Pipeline 0 |
| |                      +-------------------------+----------------+------+---------+----------------+------------------------+-----------+---------------------+
| +AllNodesScan@neo4j    | p                       | 163            | 163  | 164     | 72             | 4/0                    | 1.705     | Fused in Pipeline 0 |
+------------------------+-------------------------+----------------+------+---------+----------------+------------------------+-----------+---------------------+

1 row
```

출처: https://en-core.com/kor/board/notice?viewMode=view&ca=+tech&sel_search=&txt_search=&page=1&idx=101

IFA(Index-Free-Adjacency

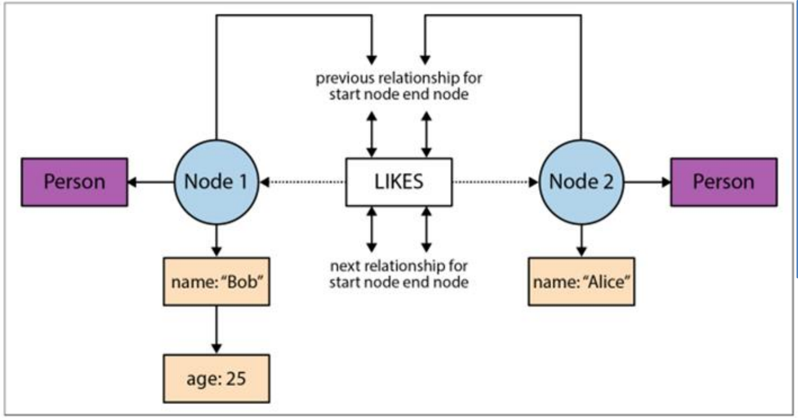
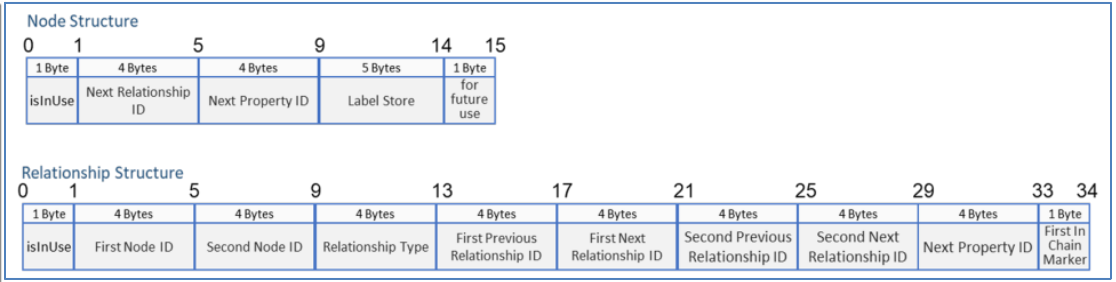


Figure 6-5. How a graph is physically stored in Neo4j




```
Store File                         | Record size    | Contents
---------------------------------------------------------------------------------------------------------------------------
neostore.nodestore.db              | 15 B           | Nodes
neostore.relationshipstore.db      | 34 B           | Relationships
neostore.propertystore.db          | 41 B           | Properties for nodes and relationships
neostore.propertystore.db.strings  | 128 B          | Values of string properties
neostore.propertystore.db.arrays   | 128 B          | Values of array properties
Indexed Property                   | 1/3 * AVG(X)   | Each index entry is approximately 1/3 of the average property value size
```

출처: https://en-core.com/kor/board/notice?viewMode=view&ca=+tech&sel_search=&txt_search=&page=1&idx=97

Architect

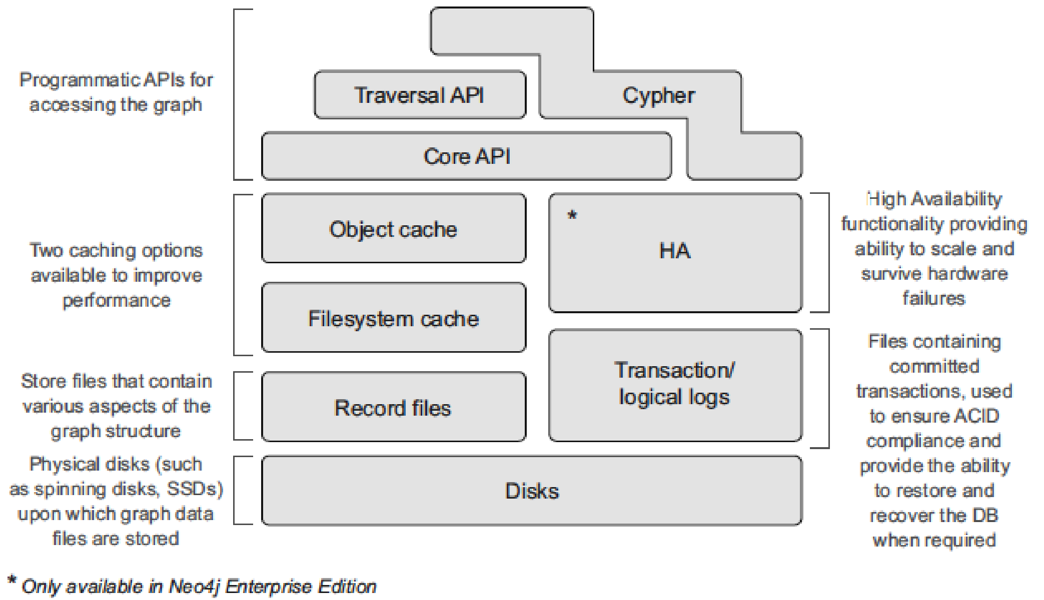
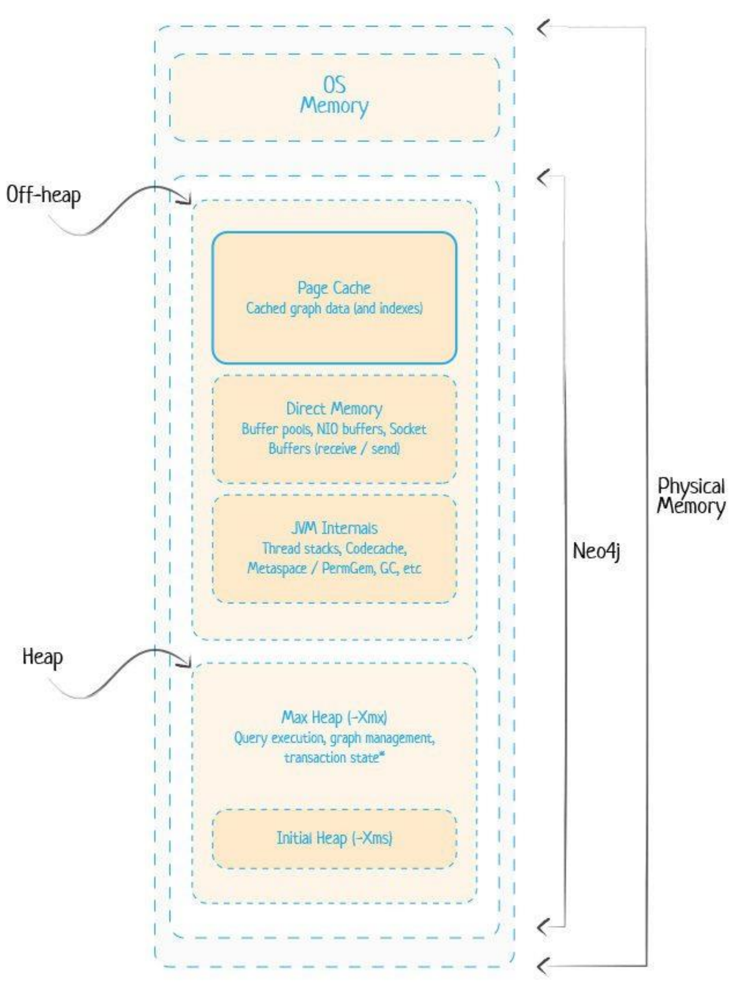


Figure 11.1 High-level overview of the Neo4j architecture



H/A Architect

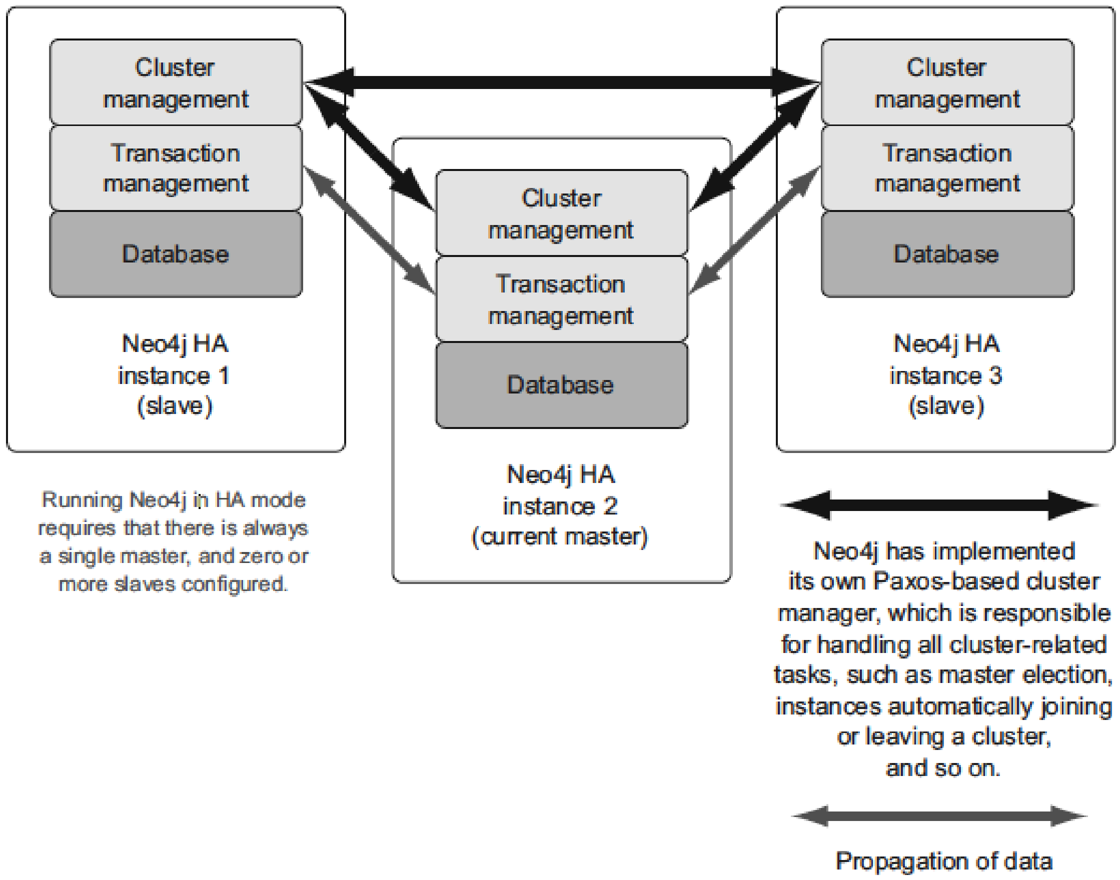
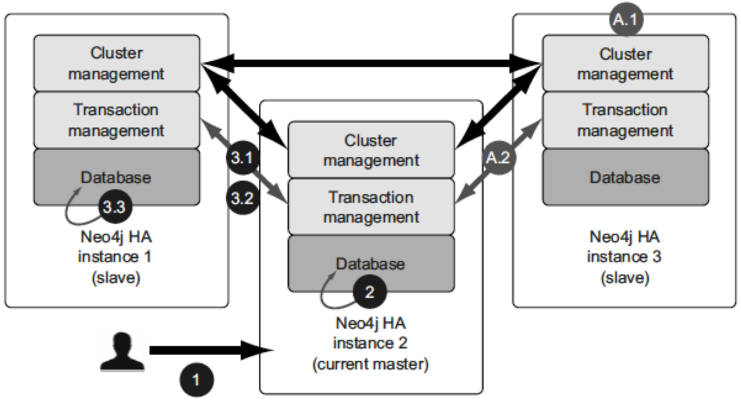


Figure 11.5 Sample Neo4j HA cluster setup with 1 master and 2 slaves

H/A Architect

- Client가 Master에게 쓰기 요청을 보냅니다



Client sends write request to a master …

1. Client issues a commit.
2. Master acquires lock and applies the transaction as per normal non-transaction rules to the master DB instance.

3.1 Master consults `ha.tx_push_factor` setting and attempts to optimistically push change to n (default 1) configured slaves—but if this fails, (2) still succeeds.

3.2 Slave ensures it is up to date first, and if not, pulls any outstanding transactions from master.

3.3 Slave applies transaction locally.

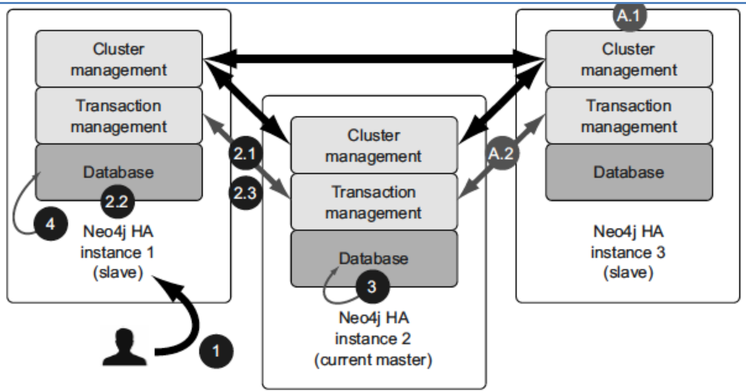
In the background …

A.1 Slave consult `ha.pull_interval` setting to determine if/when to pull updates from the master (default is not to pull updates regularly, only during write transactions).

A.2 When interval is in range, any new transactions are pulled from the master and applied locally to the slave.

Figure 11.8 Sequence of events when a write request is sent to the master instance

- Client가 Slave에게 쓰기 요청을 보냅니다



Client sends write request to a slave …

1 Client issues a commit.

2.1 Lock is acquired on master database.

2.2 Lock is acquired on slave database.

2.3 As part of its internal protocol, slave ensures it is up to date first, and if not, pulls any outstanding transactions from master.

3 Update is committed to master.

4 Provided master commit was successful, local slave transaction is also commited.

In the background …

A.1 Slave consults `ha.pull_interval` setting to determine if/when to pull updates from the master (default is not to pull updates regularly, only during write transactions).

A.2 When interval is in range, any new transactions are pulled from the master and applied locally to the slave.

Figure 11.9 Sequence of events when a write request is sent to a slave instance

Sharding

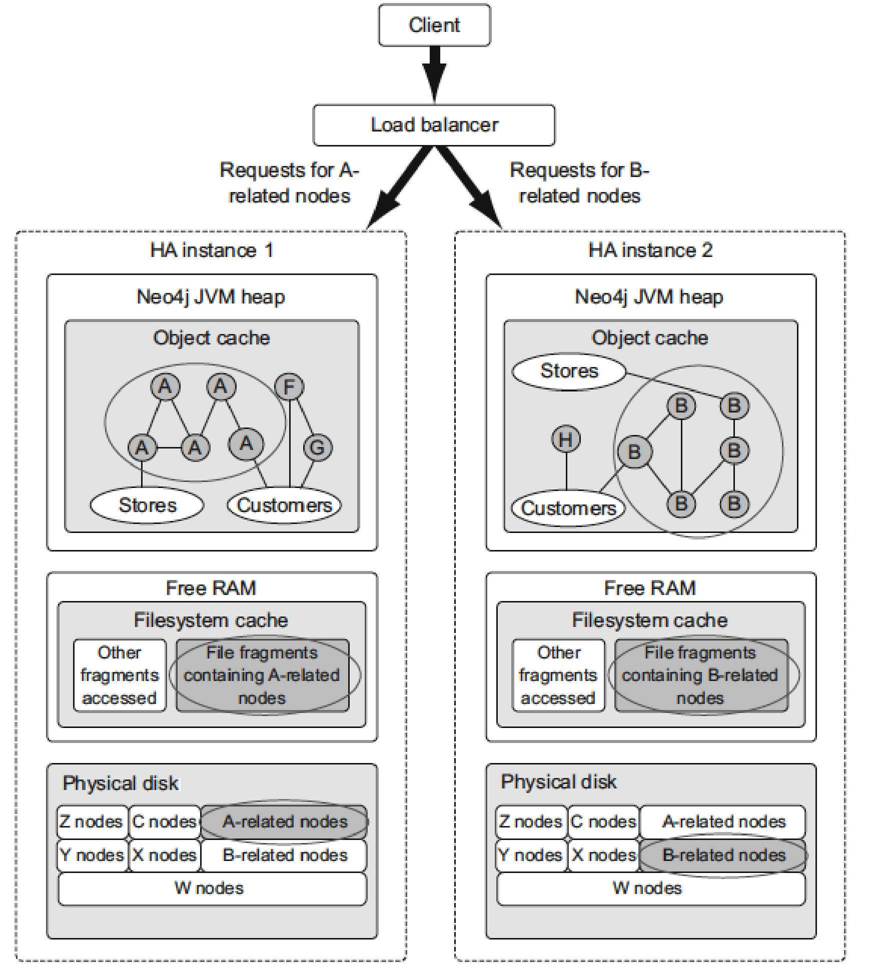


Figure 11.10 Cache sharding

TIGERGRAPH Deep Dive

Who is TigerGraph?

We provide advanced analytics on connected data

- The hyper-scalable graph database for the enterprise
- Foundational for AI and ML solutions
- Designed for efficient concurrent OLTP and OLAP workloads (HTAP)
- SQL-like query language (GSQL) accelerates time to solution
- Cloud Neutral: Google GCP. Microsoft Azure ,Amazon aws

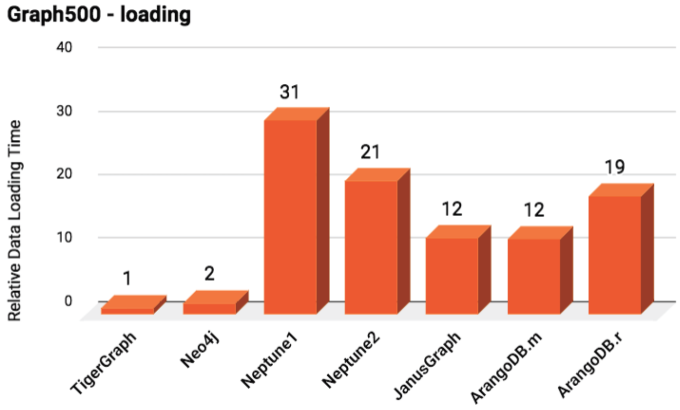
Our customers include:

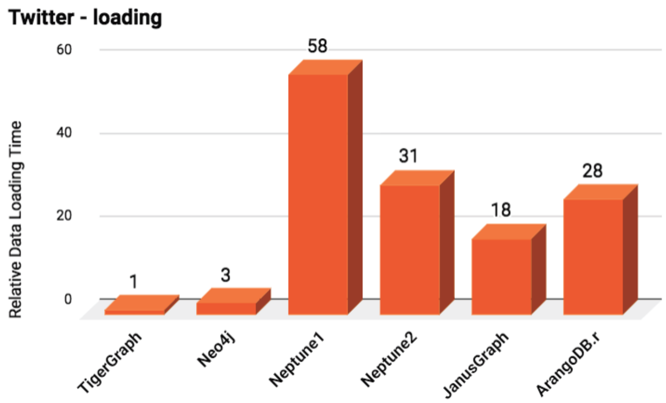
- The largest companies in financial, healthcare, telecoms, media, utilities and innovative startups in cybersecurity, and ecommerce.

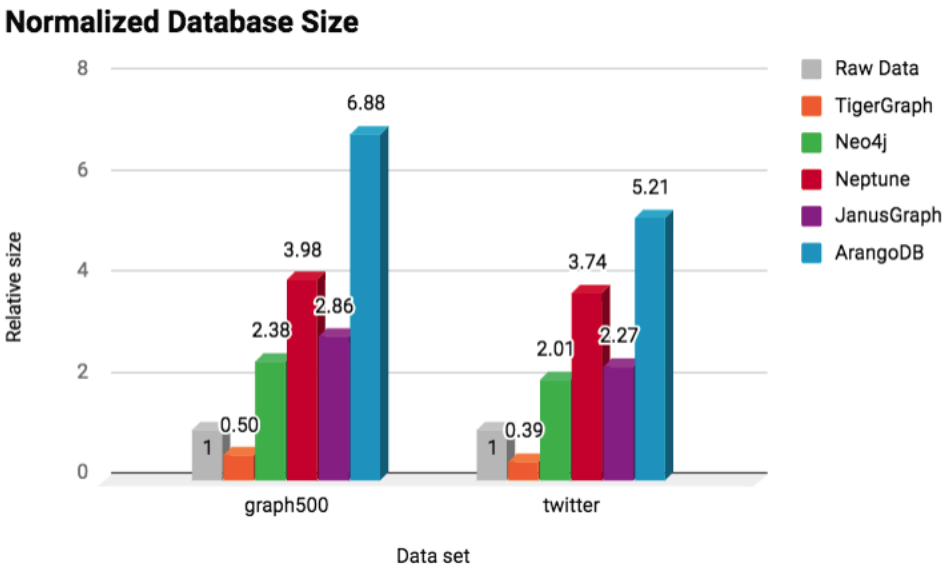
https://www.crunchbase.com/organization/tigergraph

Graph-Powered Analytics and Machine Learning with TigerGraph

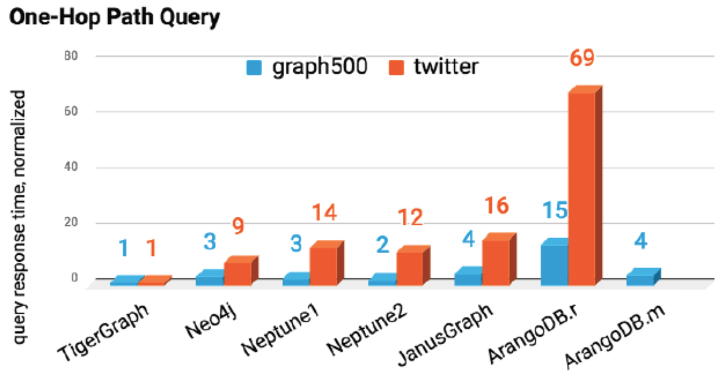
Data Loading Time and Speed, Size

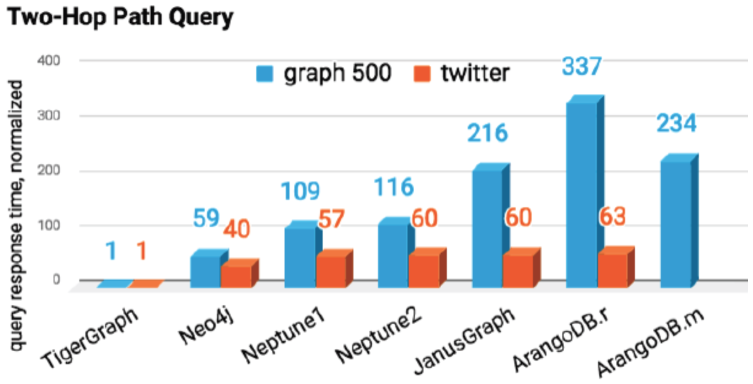


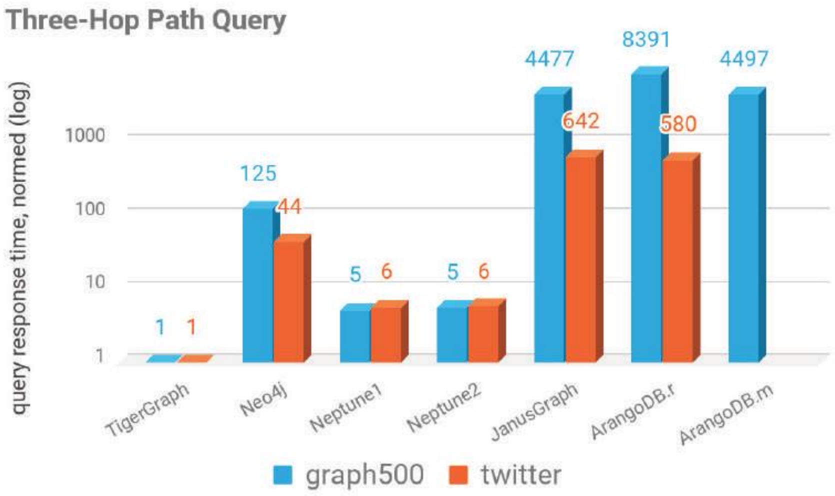




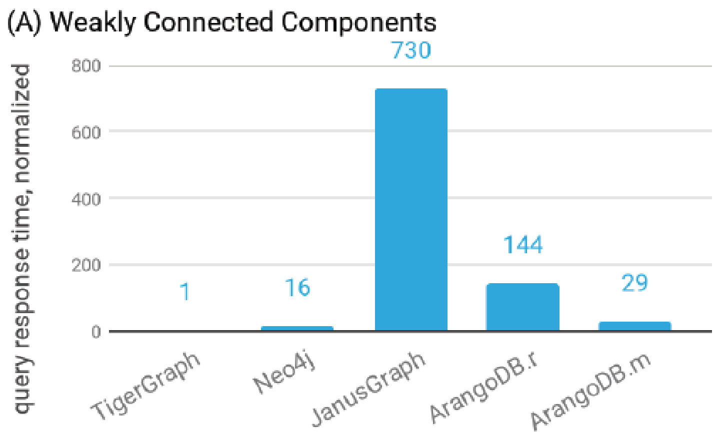
K-Neighborhood Query Time

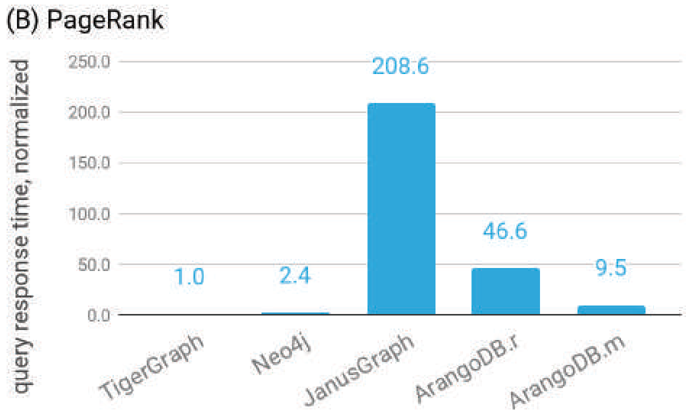


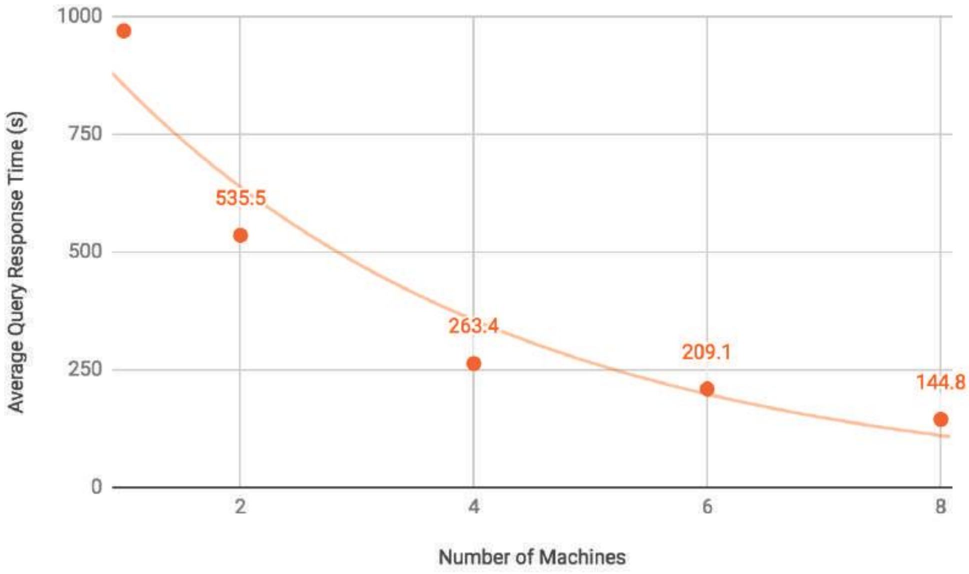




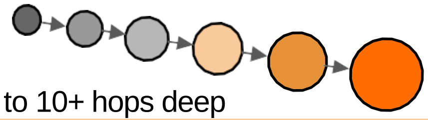
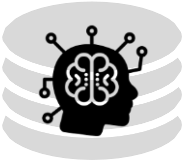
Weakly Connected Component and PageRank Queries Time



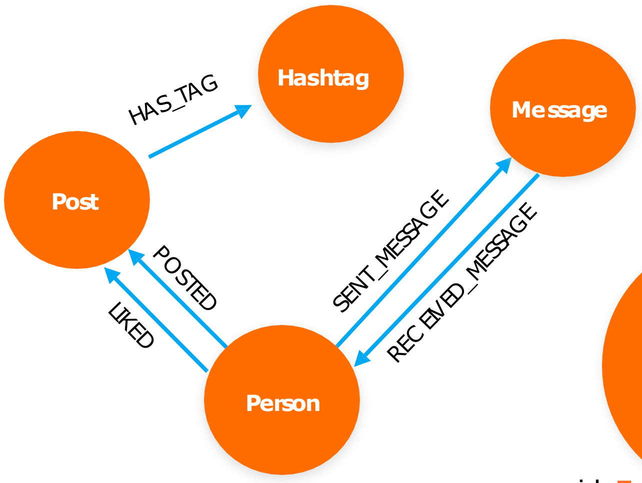




The TigerGraph Difference

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

Property Graphs - Types and Properties

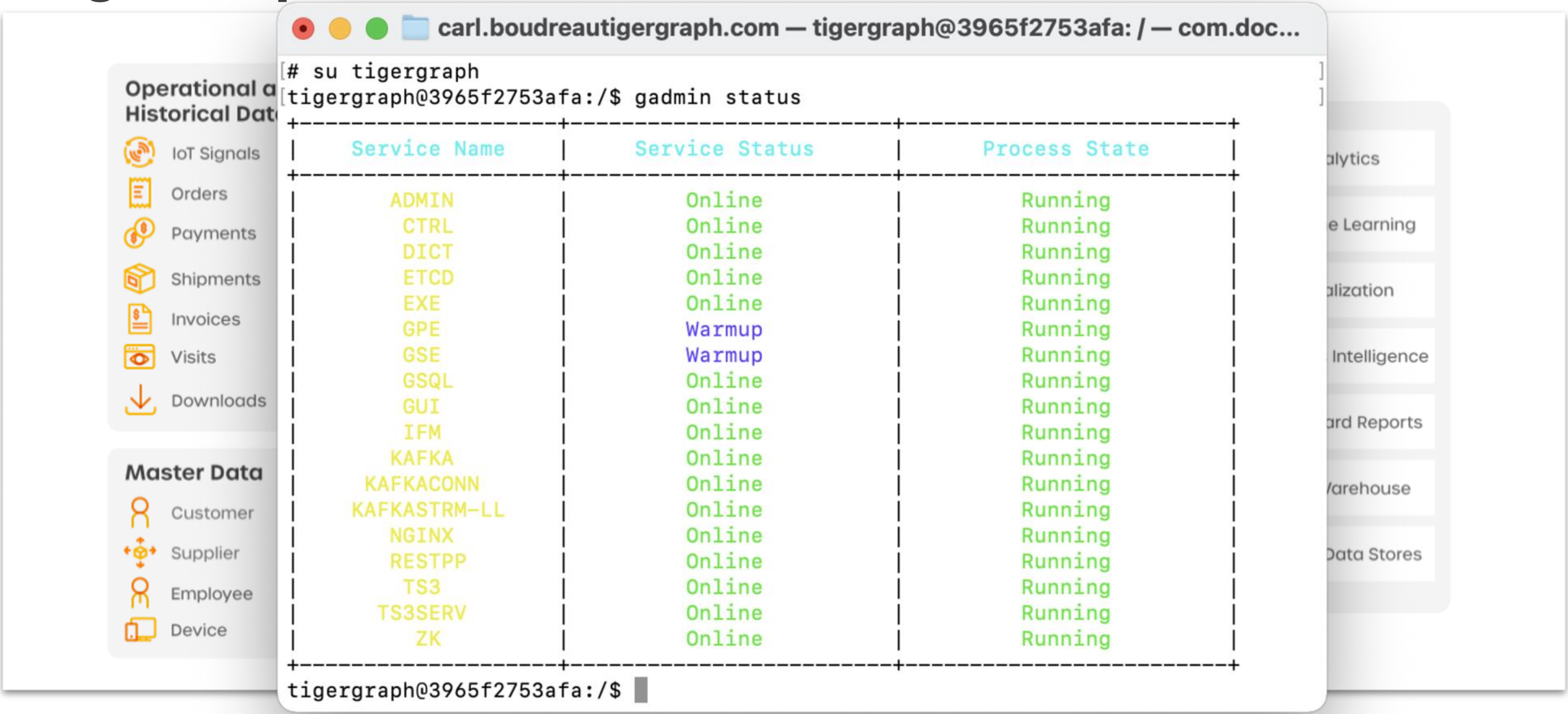


id: 1025
by_user: 19
liked_post: 12 liked_date: 2021/07/11

id: 7
email: vbarracks6@utexas.edu
username: vbarracks6
name: Virgie Barracks
join_date: 2021/09/02

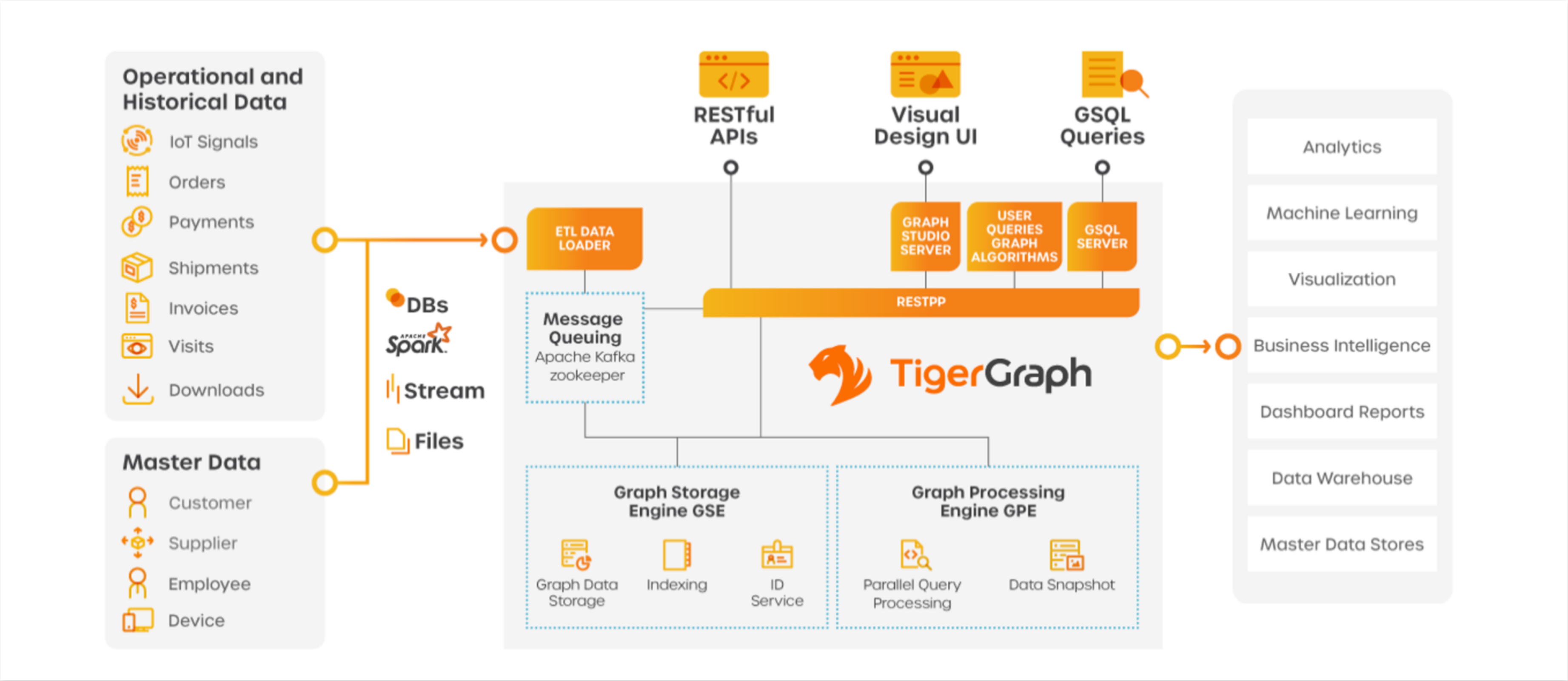
id: 12
content: Suspendisse ornare ….
posted_date: 2020/07/31
posted_by: 7
deleted: False

TigerGraph Architecture

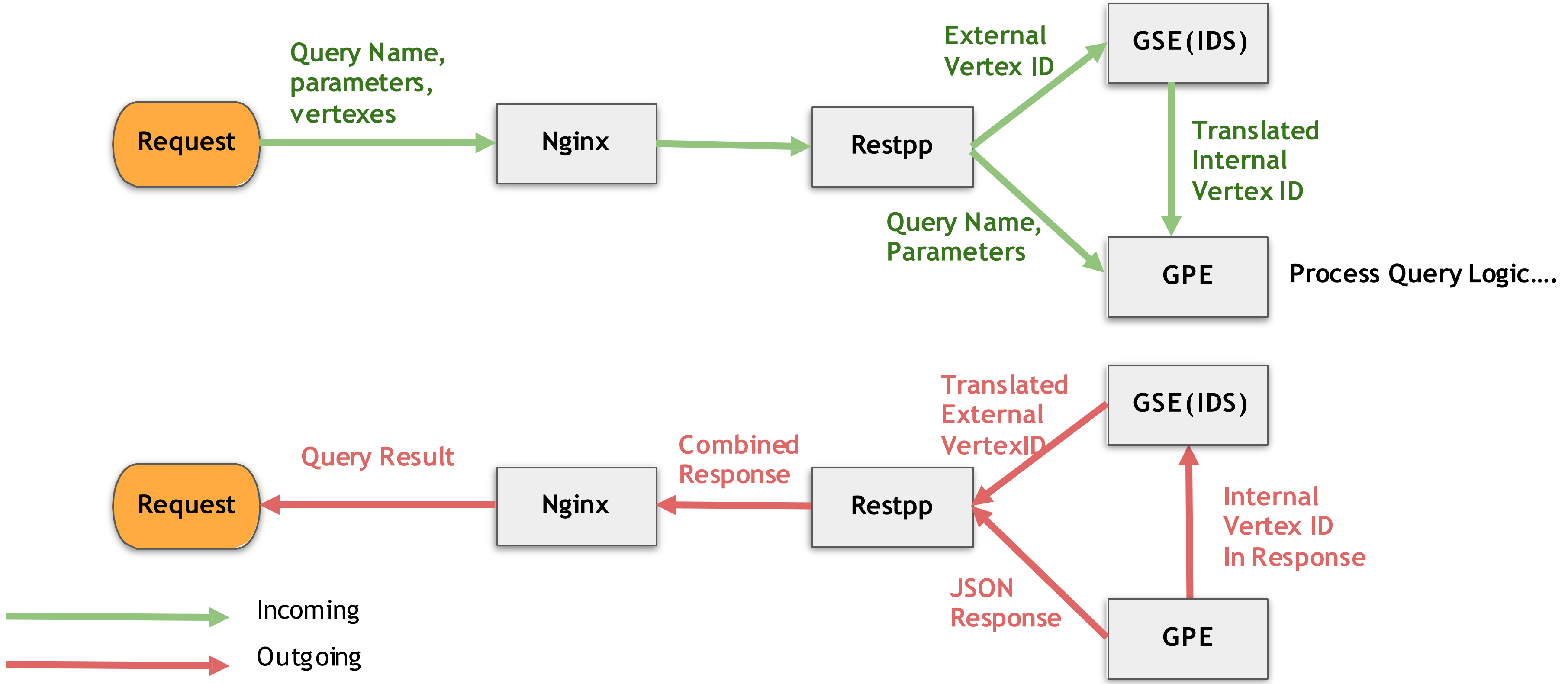



```
# su tigergraph
tigergraph@3965f2753afa:/$ gadmin status
+--------------------+-------------------------+-------------------------+
|    Service Name    |     Service Status      |      Process State      |
+--------------------+-------------------------+-------------------------+
|       ADMIN        |         Online          |         Running         |
|        CTRL        |         Online          |         Running         |
|        DICT        |         Online          |         Running         |
|        ETCD        |         Online          |         Running         |
|        EXE         |         Online          |         Running         |
|        GPE         |         Warmup          |         Running         |
|        GSE         |         Warmup          |         Running         |
|        GSQL        |         Online          |         Running         |
|        GUI         |         Online          |         Running         |
|        IFM         |         Online          |         Running         |
|       KAFKA        |         Online          |         Running         |
|     KAFKACONN      |         Online          |         Running         |
|   KAFKASTRM-LL     |         Online          |         Running         |
|       NGINX        |         Online          |         Running         |
|       RESTPP       |         Online          |         Running         |
|        TS3         |         Online          |         Running         |
|      TS3SERV       |         Online          |         Running         |
|         ZK         |         Online          |         Running         |
+--------------------+-------------------------+-------------------------+
tigergraph@3965f2753afa:/$
```

TigerGraph Architecture



Query Processing workflow



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.
- No Duplicates

TigerGraph Native Graph Storage

“USER123” <---> 1234321

IDS: Bidirectional external ID to Internal ID mapping

1234321, John, 33, john@abc.com
1234322, Tom, 27, tom@abc.com
...

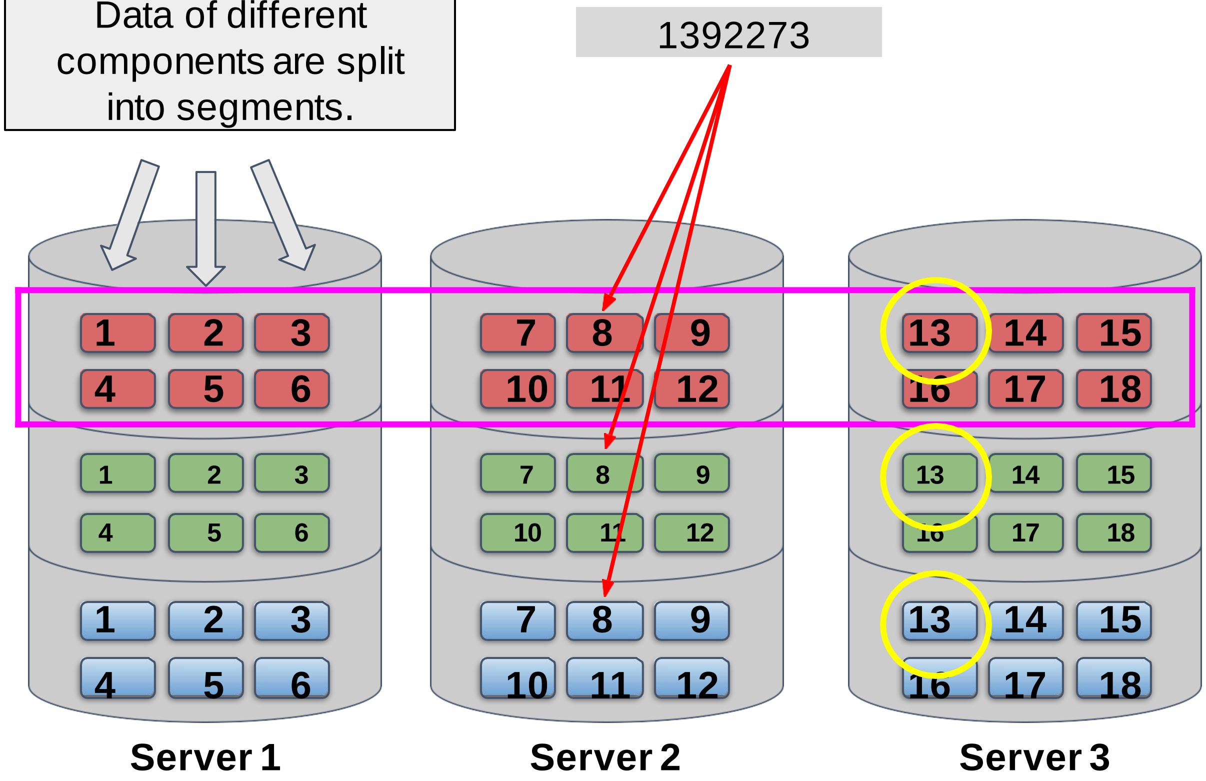
Vertex Partitions: Vertex internal ID and attributes

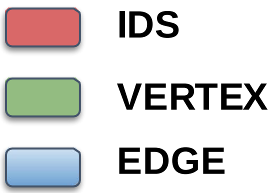
1234321, 1234322, 2020-04-23, 3.3
1234321, 1234324, 2020-02-13, 2.3
...

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

Distributed Native Graph Storage

Data of different components are split into segments.



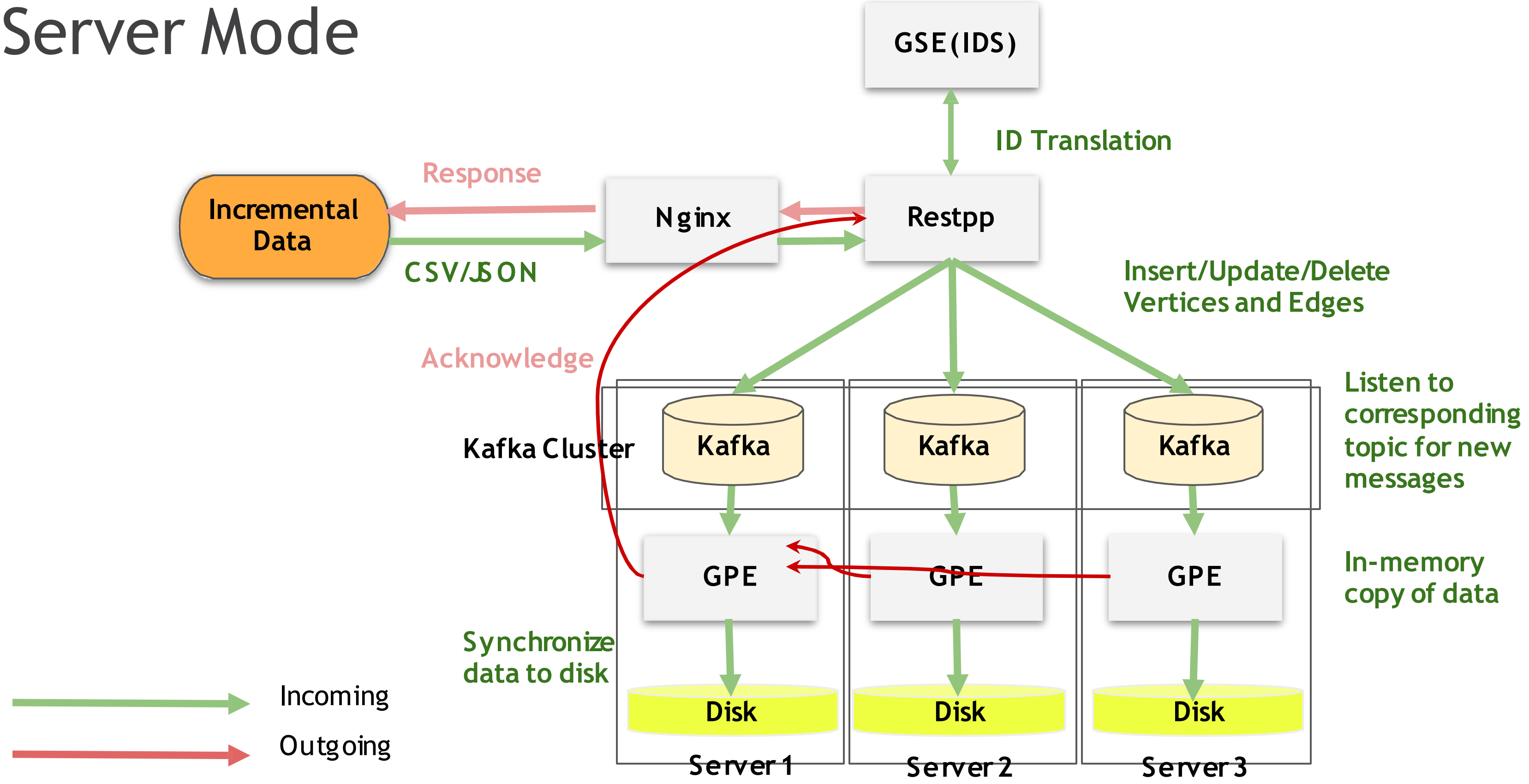


The segments are stored distributedly across the cluster.

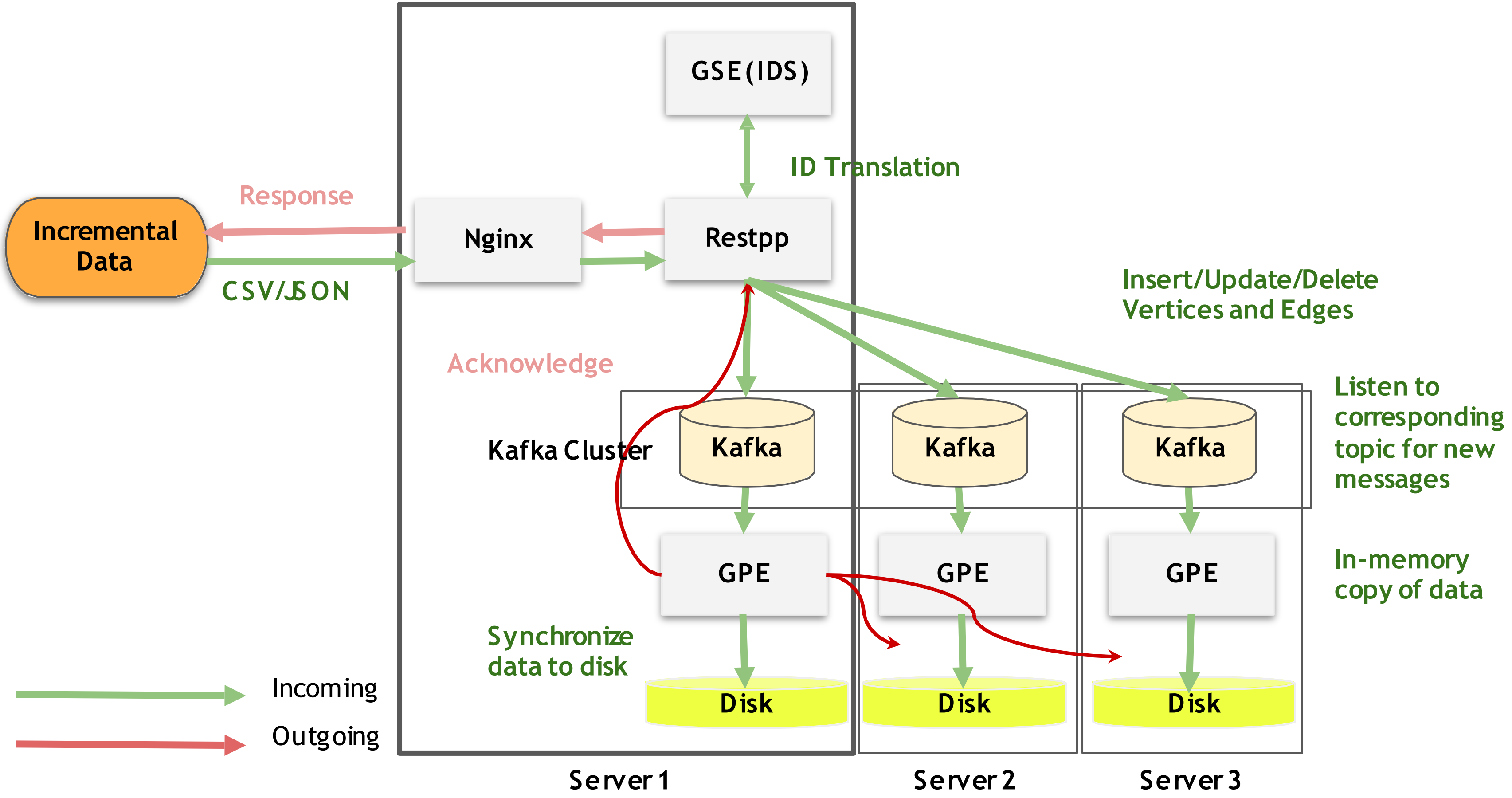
The segments of different components with same ID stores data for the same set of vertices under the same vertex type.

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

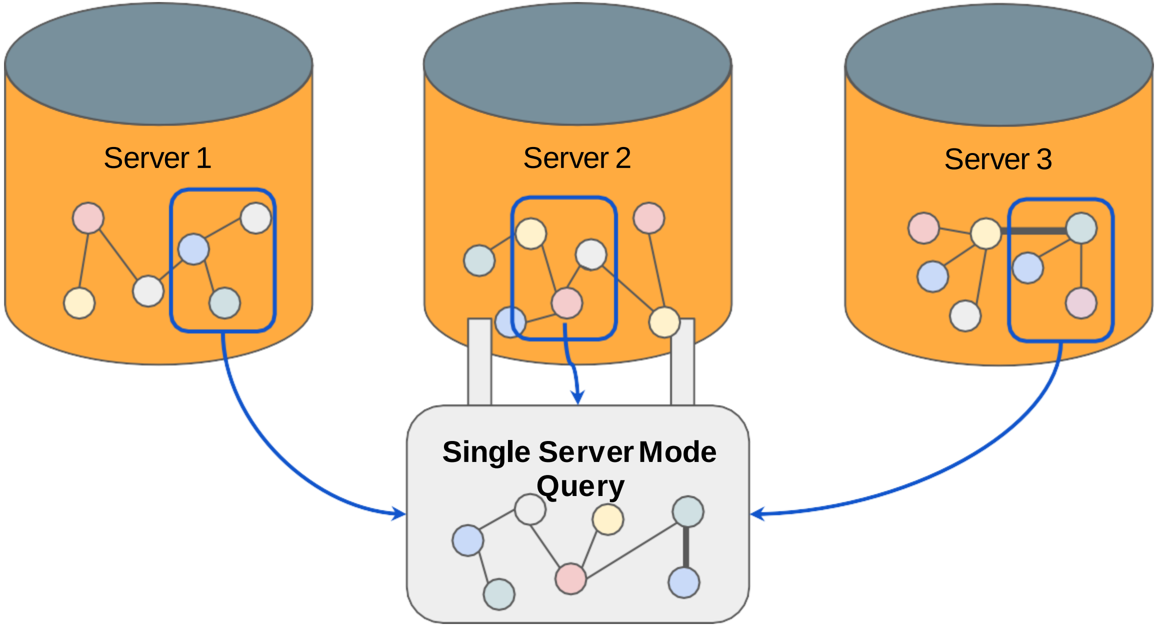
Data Ingestion in Distributed Cluster in Distributed Server Mode



Data Ingestion in Distributed Cluster in Single Server Mode



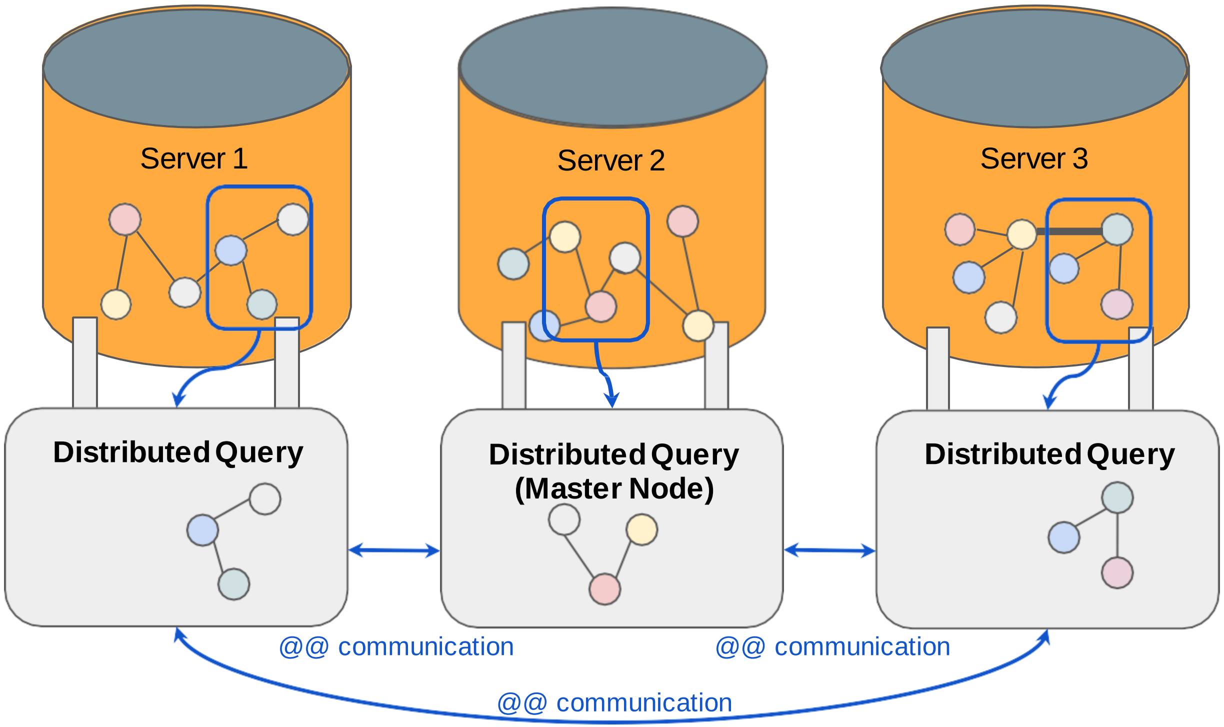
MPP - Distributed Cluster in Single Server mode



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.

MPP - Distributed Cluster in Distributed mode

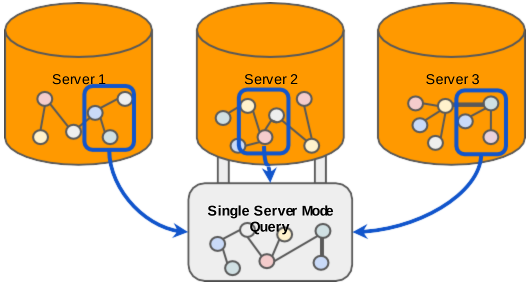


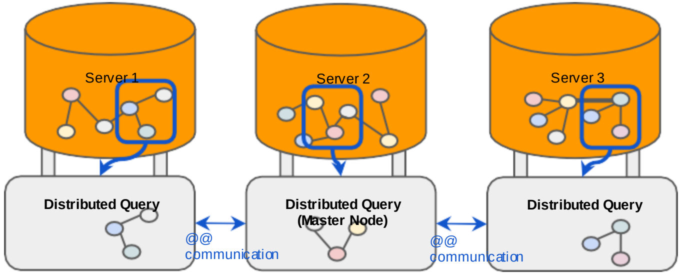
Distributed Mode

- 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.

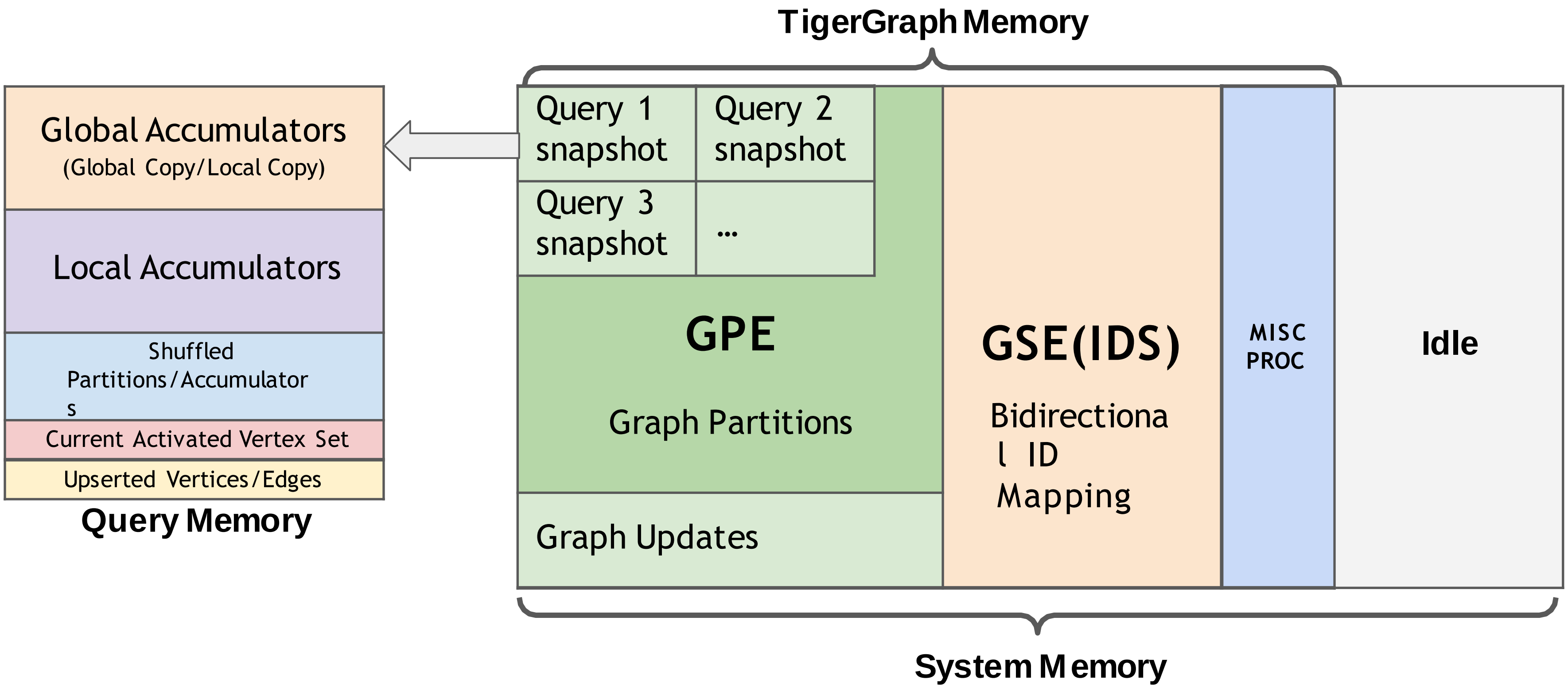
Ex: Point query, single entity-based transaction/update

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.)

TigerGraph Memory Usage Overview



Data Encryption & Data Compress

- 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

- Compressed Data
  - Can Compress Data
  - Lz4, Snappy, etc

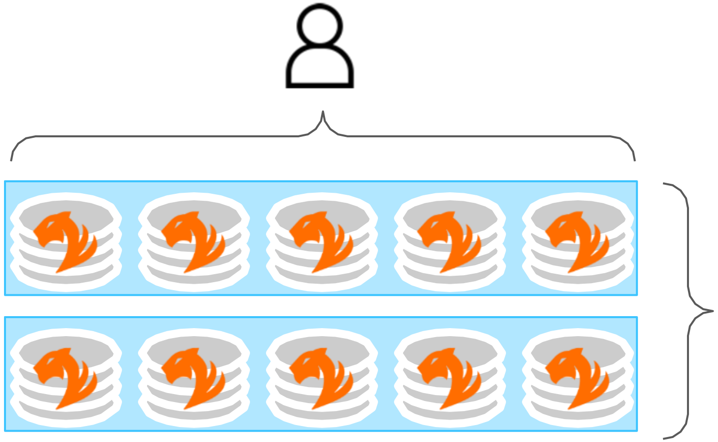
NON-FUNCTIONAL FEATURES

- High Availability
- Access Control & Security
- Transaction Management

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)

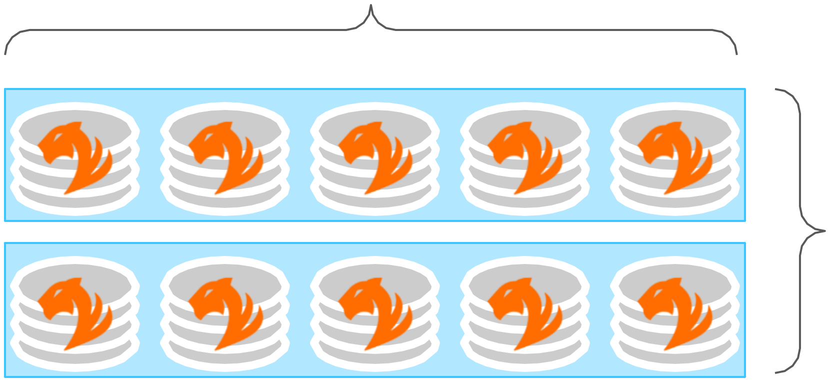
- write to all
- read from any
- strong consistency

Advantages:

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

High Availability

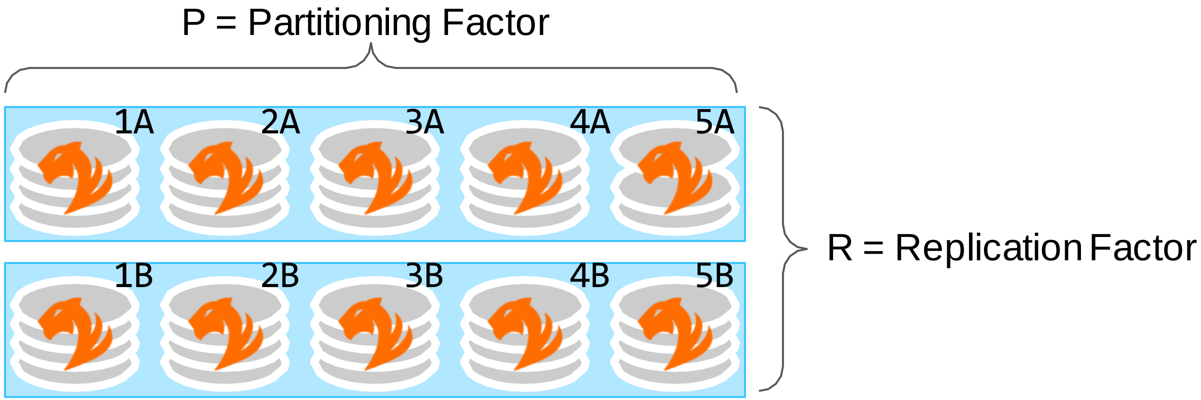
- TigerGraph HA Replication provides both Increased Throughput and Continuous Operation
- Cluster size = P X R (Partitions x Replicas)
- Any cluster size is allowed, except 1x2



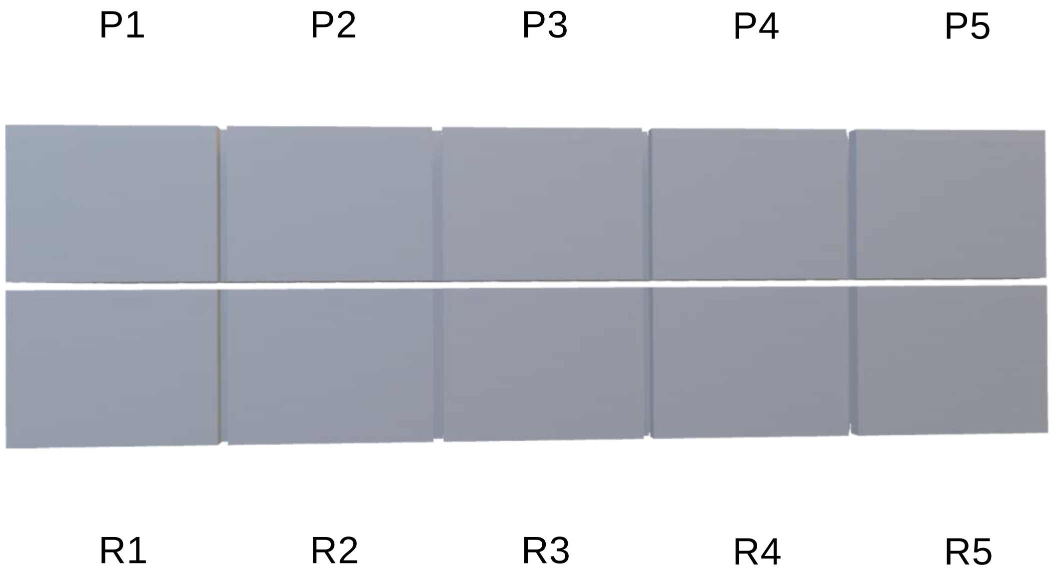
HA and Concurrency

- 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.

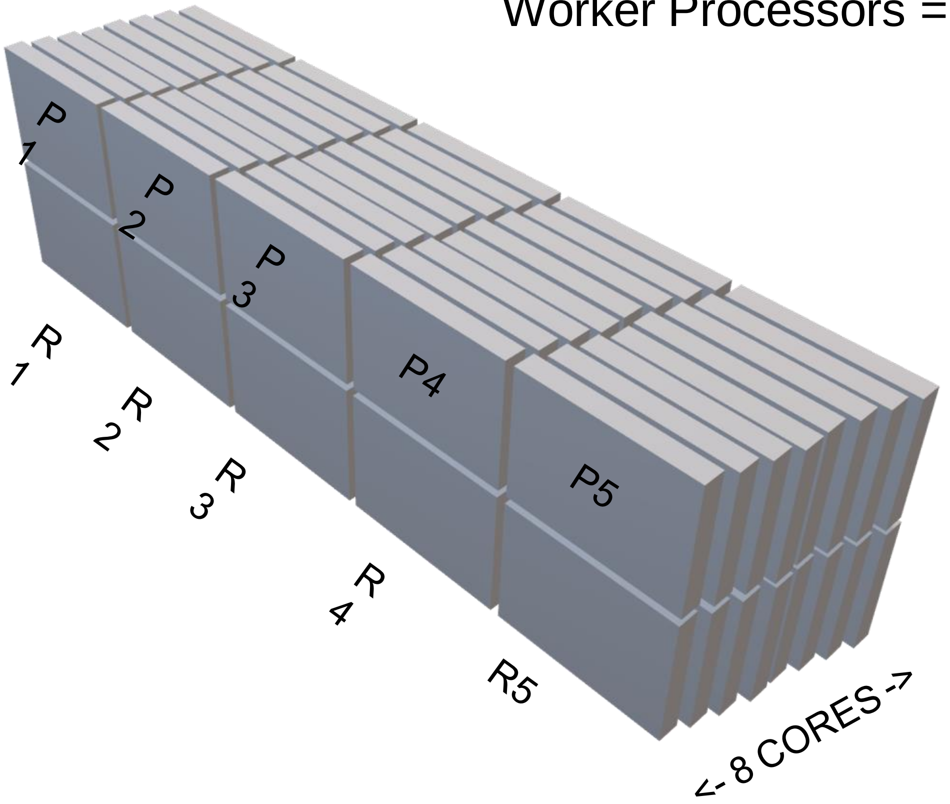


HA and Concurrency



HA and Concurrency

Worker Processors = 8x5x2 = 80



HA and Distributed Storage

HA cluster

An HA cluster needs at least 3 server machines, even if the system only has one graph partition.

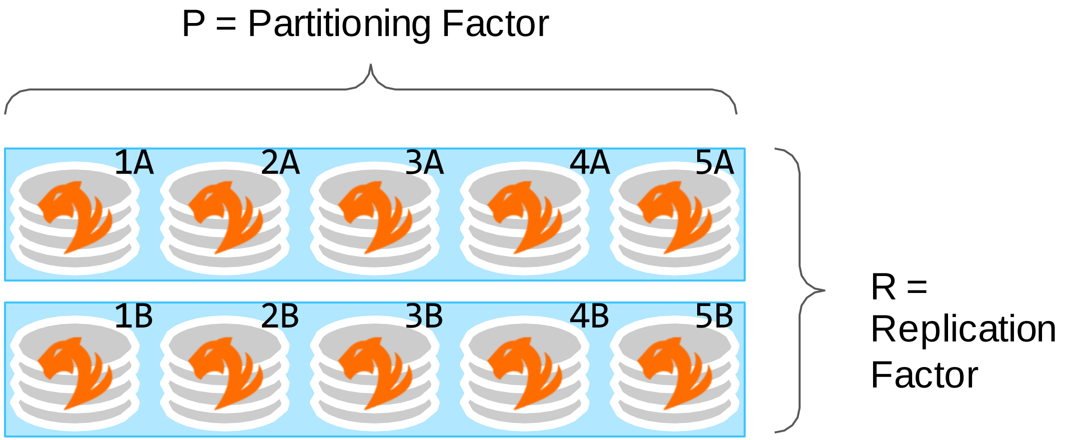
Machines can be physical or virtual.

Distributed System

For a distributed system with N partitions (where N > 1), the system must have at least 2N machines.

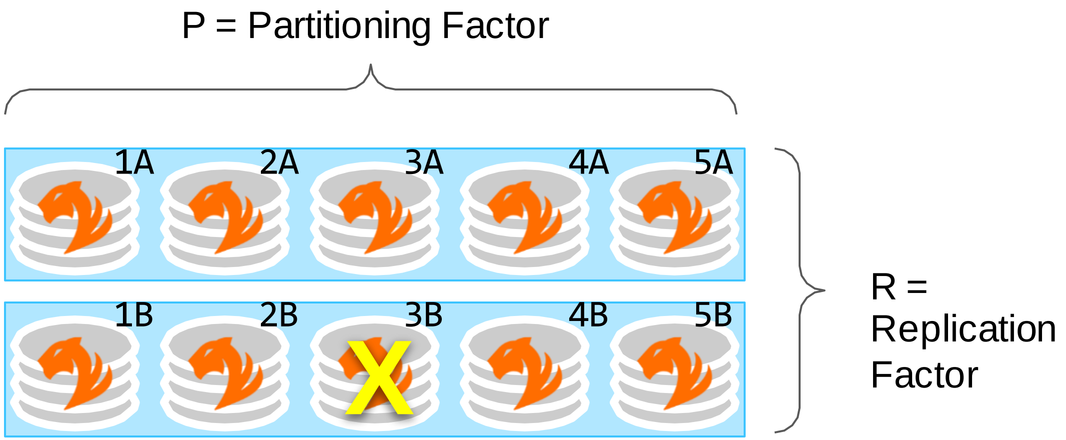
HA Read and Write Behavior

- 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.)



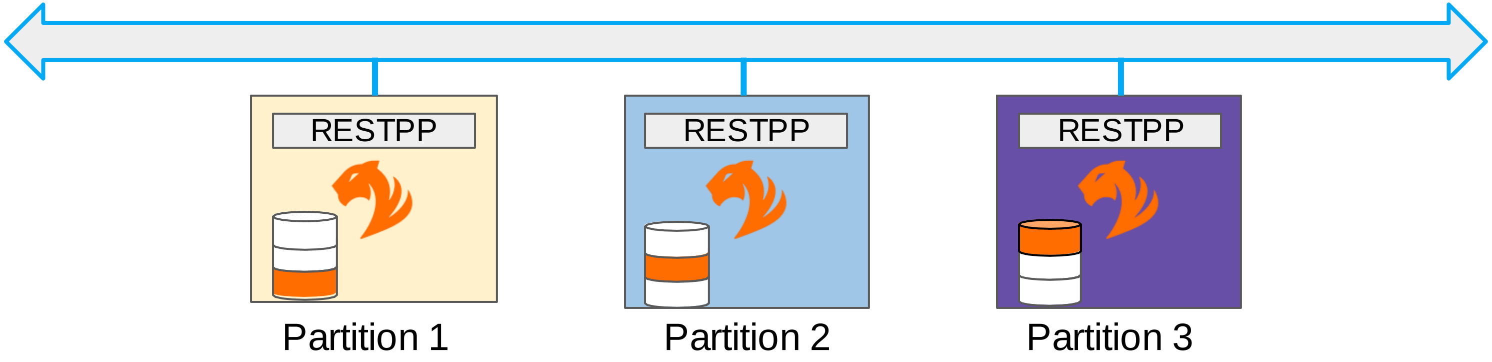
HA Continuous Operation

- 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.



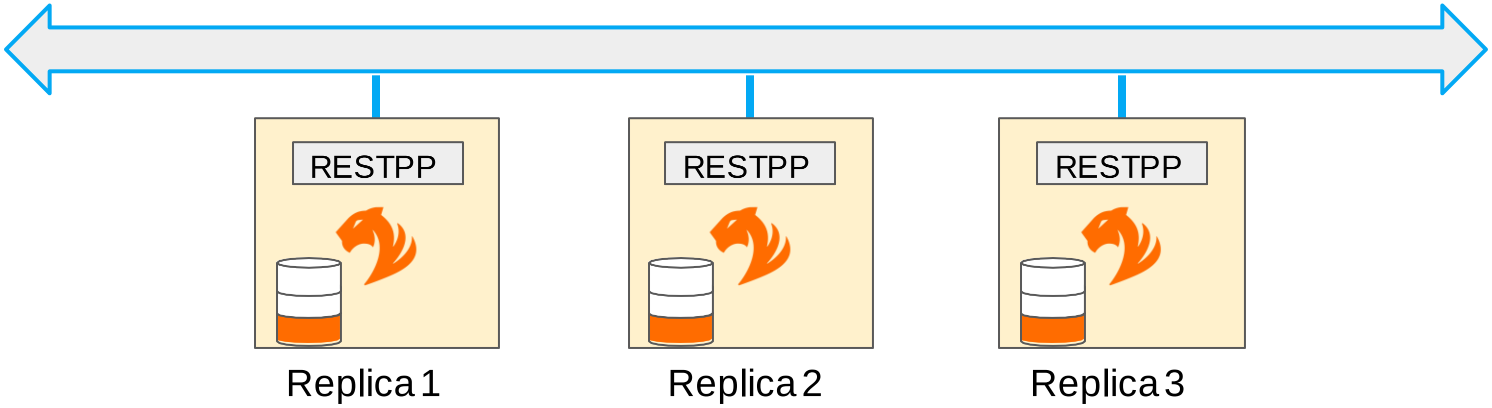
Distributed Data for Massive Datasets

- Graph DB is partitioned across multiple server nodes.
  - Default partitioning scheme: uniform hash for load balancing
- RESTPP acts as scheduler and distributor.
- For ACID:
  - Transactions are not committed until all partitions are updated → Strong Consistency



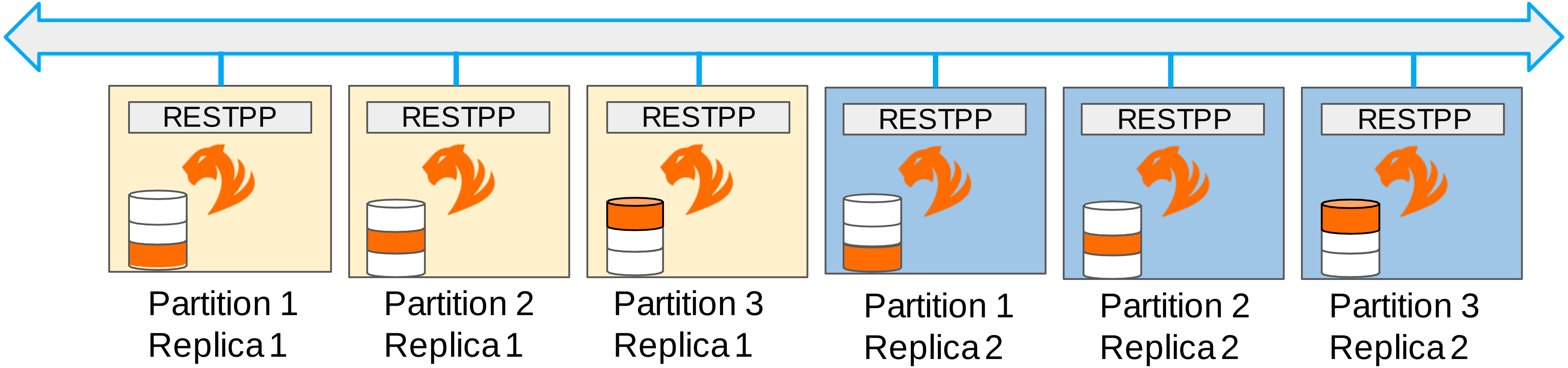
HA, without External Load Balancing

- One RESTPP is chosen as the master, for load balancing decisions.
- Default scheduling scheme is round robin.



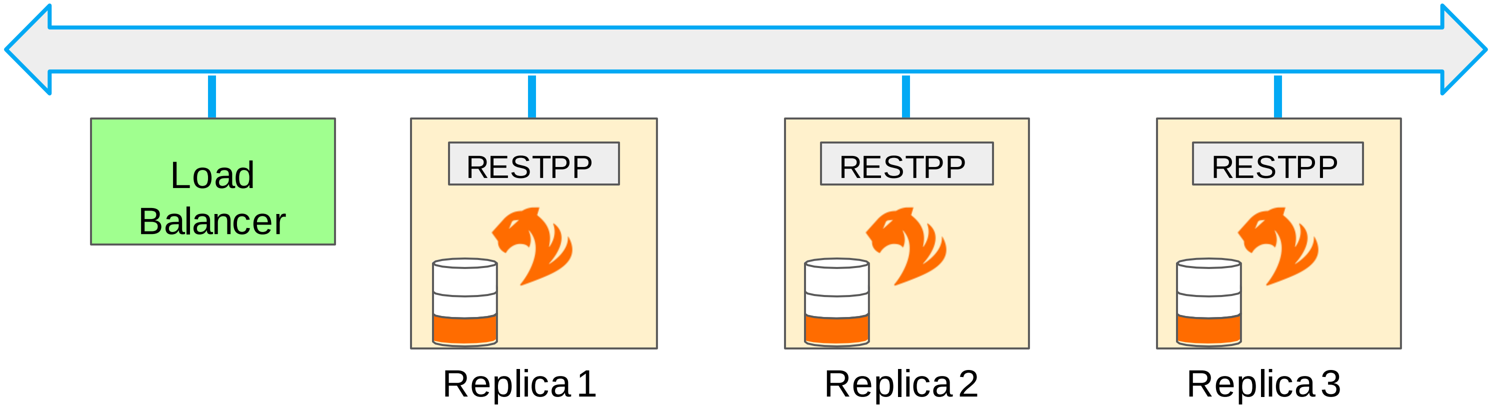
HA and Distributed Hybrid Storage

- System has both distributed and HA storage



HA, with External Load Balancing

- User selects an external load balancing component/service.



TigerGraph

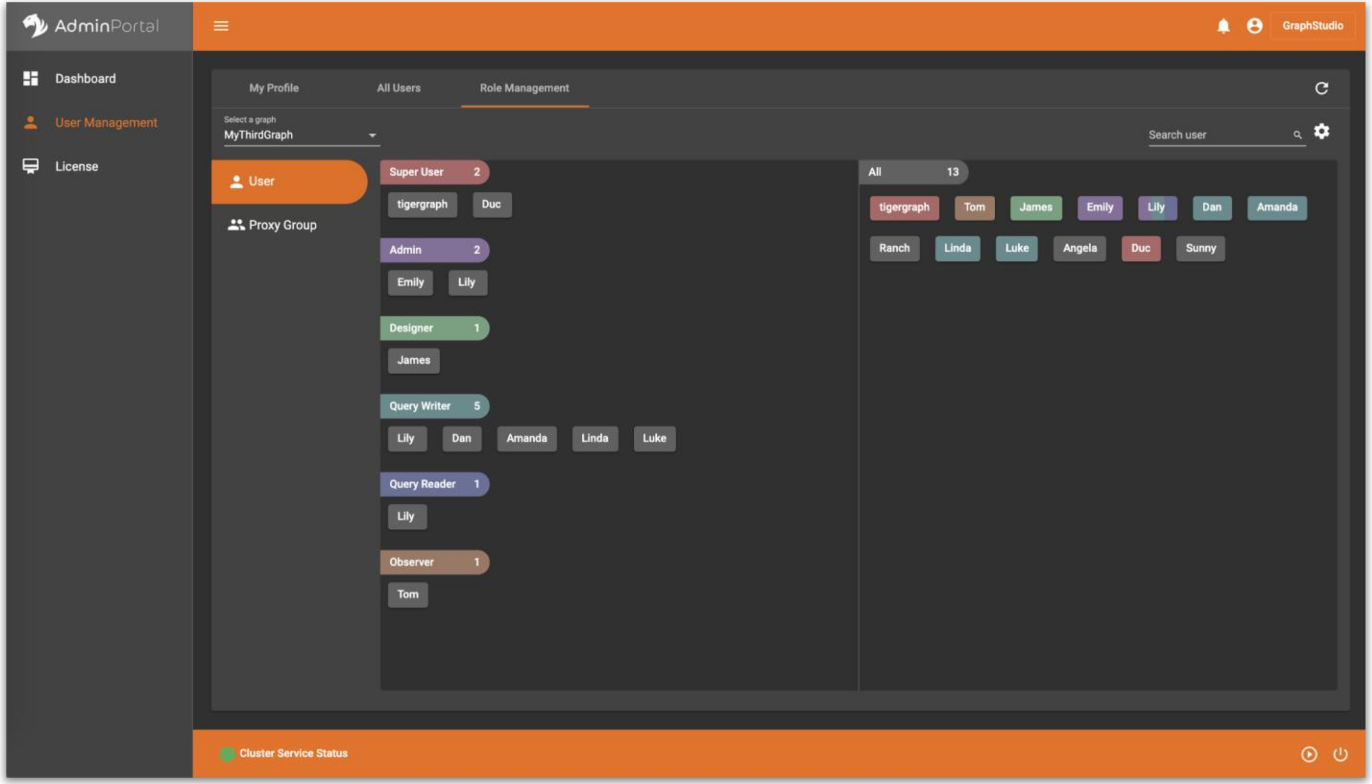
NON-FUNCTIONAL FEATURES

- High Availability
- Access Control & Security
- Transaction Management

Role-Based Access Control

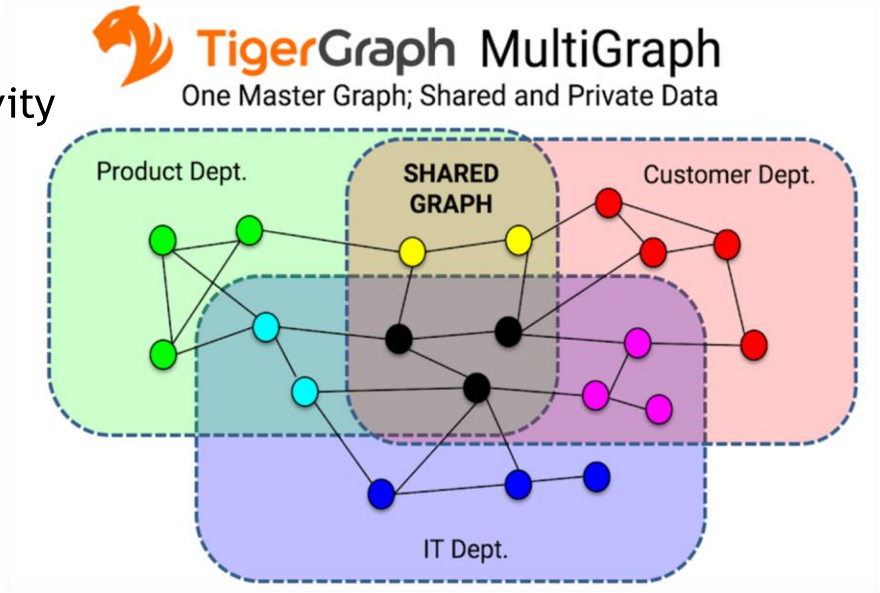
- Follows SQL approach for roles.
  GSQL:
  ```
  GRANT <role> ON GRAPH <graph> TO <user1, user2, … >
  REVOKE <role> ON GRAPH <graph> FROM <user1, user2, … >
  ```
- Can map TigerGraph roles to external LDAP roles and groups.

Admin Portal UI for Managing User Privileges



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:

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

User-Defined Roles in version 3.2 above

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	-	-

TigerGraph

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
A transaction with update operations may insert/delete multiple vertices/edges or update the attribute values of multiple edges/vertices. Such update requests 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.**

GSQL Queries

- Exploring data using GraphStudio can be interesting, but there are limitations
- GSQL queries however give the most flexibility when interacting with a graph
- A GSQL query is a user defined procedure
  - There can be one or more input parameters
  - It can produce data in two ways, by returning a value or by “printing”

Query Running Modes

Queries can be run in **Interpret Mode** or as **Installed** queries

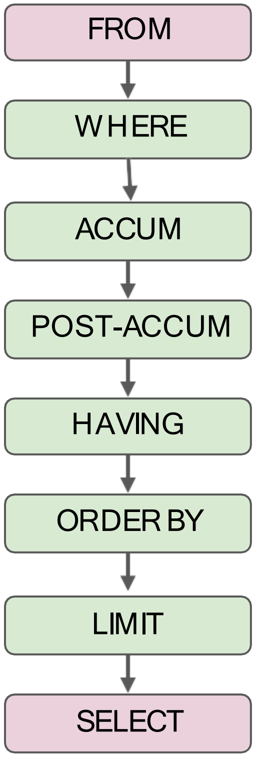
- Interpret mode does not require the query to be compiled or installed, the trade off is that an interpreted query is not as efficient as an installed query. There are also some limitations in functionality for queries run in this mode

- Installed queries have no such limitations and become accessible as reachable endpoints

The Basics

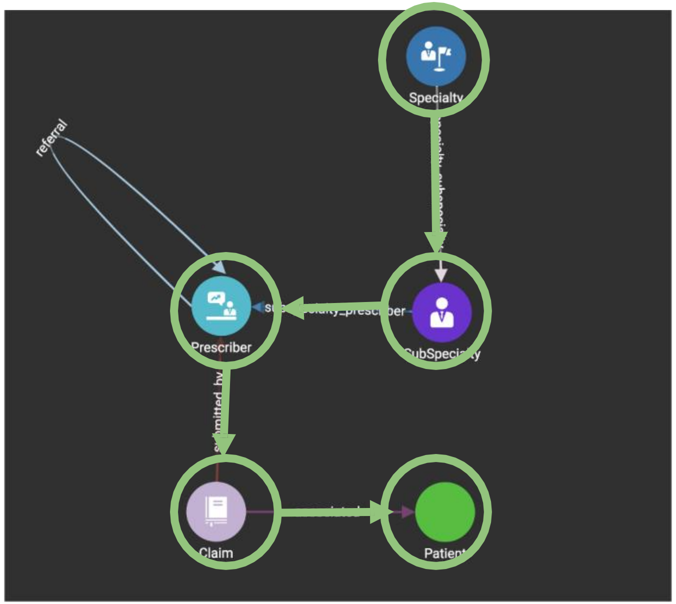
```
resultSet = SELECT  vSet
   FROM ( edgeSet | vertexSet )
   [whereClause] [accumClause]
   [postAccumClause] [havingClause]
   [orderClause] [limitClause]  ;
```

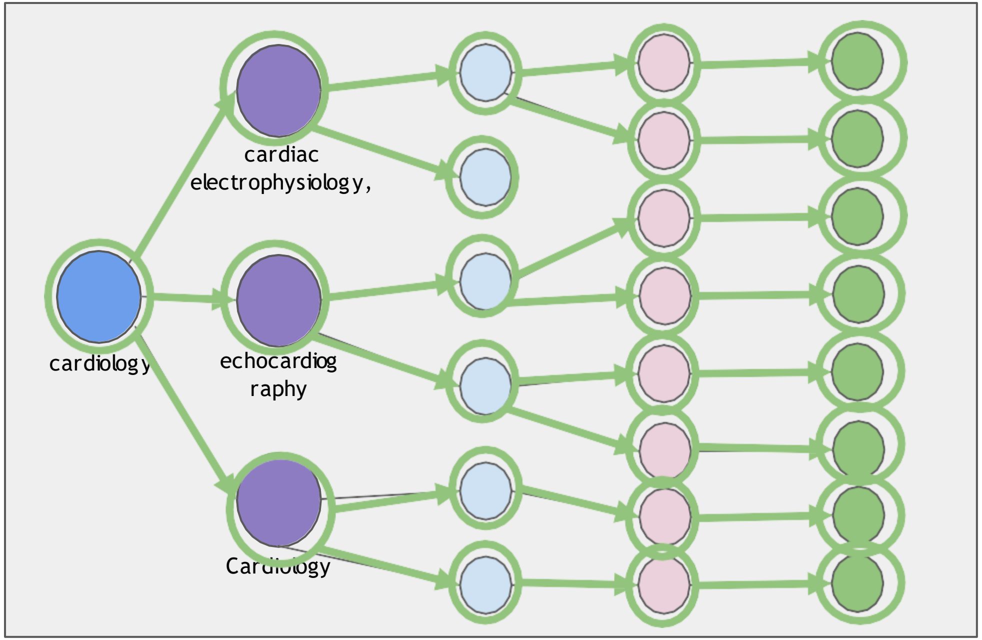
- **FROM**: select active vertices & edges.
- **WHERE**: conditionally filter the active sets
- **ACCUM**: iterate on edge set; compute with accumulators
- **POST-ACCUM**: iterate on vertex sets; compute with accumulators
- **HAVING**: conditionally filter the result set
- **ORDER BY**: sort
- **LIMIT**: max number of items
- **SELECT**: result from source or target set



The Basics

GSQL traverses the graph from one set of vertices, through selected edges originated from the starting set, to another set of vertices:





The Basics

```
CREATE QUERY getKhopNeighbor(int k, vertex input) FOR GRAPH MyGraph {

  OrAccum<BOOL> @visited;
  ListAccum<EDGE> @@edgeList;

  start = {input};

  WHILE start.size() > 0 limit k DO
    start = SELECT t from start-(:e)-:t
            WHERE t.@visited == false
            ACCUM @@edgeList += e
            POST-ACCUM t.@visited = true;
  END;

  print @@edgeList;
}
```

Data Declare Session

Query Logic Session

The Basics

The query always starts with a seed vertex set - that logic originates from

Start from a single vertex:

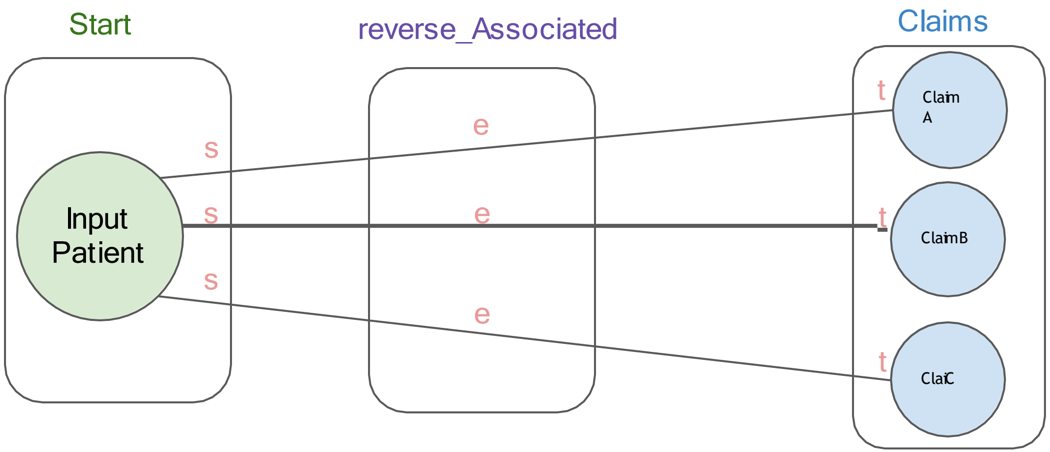
```
CREATE QUERY example(VERTEX input_ver) FOR GRAPH g {
    start_set = {input_ver};
    ….
}
```

- Here start_set is the variable name of vertex set variable
- A vertex set variable is where a SELECT statement starts from
- A vertex set variable is also the outcome of a SELECT statement
- In this case the vertex set start_set only contains one single vertex that is the input parameter
- This is the recommended way to start your traversal logic

The Basics

Find all the claims of a patient

```
CREATE  QUERY GetClaims(vertex<User> input_patient) FOR GRAPH Social {
   Start = {input_patient};
   Claims = SELECT  t FROM Start:s-(reverse_Associated:e)-Claim:t;
   PRINT Claims;
}
```



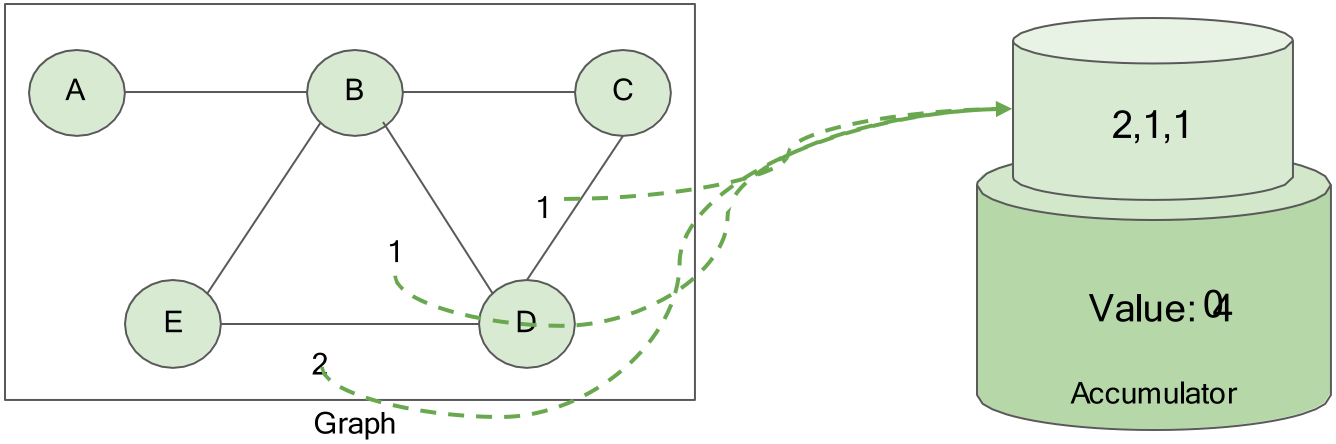
- Start is a vertex set initialized by the input vertex input_patient
- FROM clause finds edges which match the pattern: source vertex is in Start, edge type is reverse_Associated, and target vertex is restricted to User type.
- For each edge satisfies the conditions in the FROM clauses, s,e and t are aliases of source vertex, edge and target vertex.
- s, e and t are not keywords, you can rename them.
- Claims is a new vertex set equal to t.

Accumulators

Accumulators are special type of variables that accumulate information about the graph during the traversal

Accumulating phase 1: receiving messages, the messages received will be temporarily put to a bucket that belongs to the accumulator

Accumulating phase 2: The accumulator will aggregate the messages it received based on its accumulator type - the aggregated value will become the accumulator's value, and its value can be accessed



Accumulators

For example:

The teacher collects test papers from all students and calculates an average score.

Teacher: accumulator

Student: vertex/edge

Test paper: message sent to accumulator

Average Score: final value of accumulator

Phase 1: teacher collects all the test paper

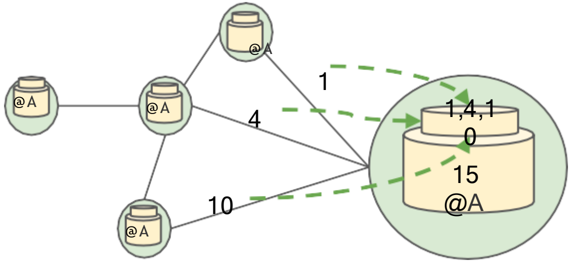
Phase 2: teacher grades it and calculate the average score.

Accumulators

Local Accumulators:

- Each selected vertex has its own accumulator
- Local means per vertex - each vertex does its own processing and considers what it can see/read/write

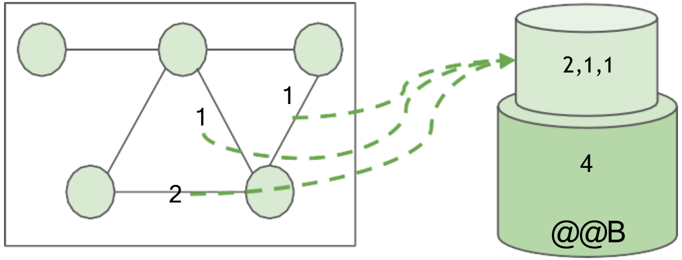
e.x. SumAccum @A;



Global Accumulators:

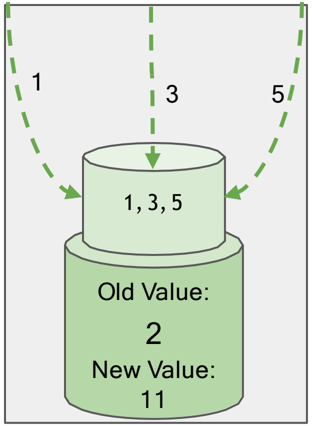
- Stored in stored globally, visible to all
- All vertices and edges have access

e.x. SumAccum @@B;



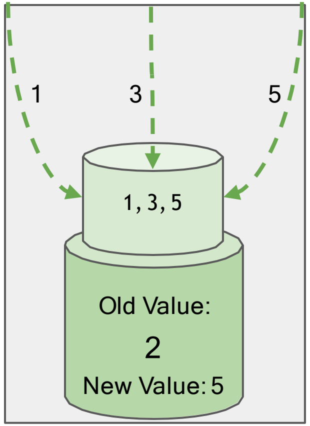
Accumulators

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



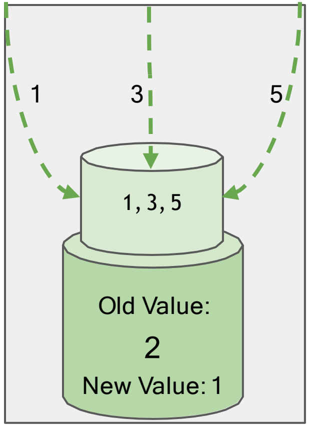
SumAccum<int>

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



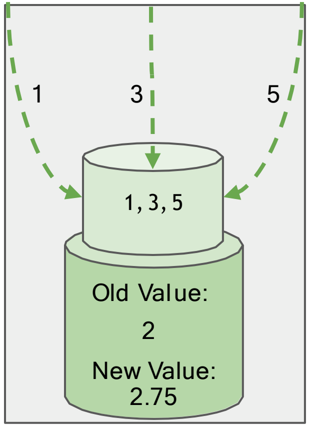
MaxAccum<int>

The MaxAccum types calculate and store the cumulative maximum of a series of values.



MinAccum<int>

The MinAccum types calculate and store the cumulative minimum of a series of values.

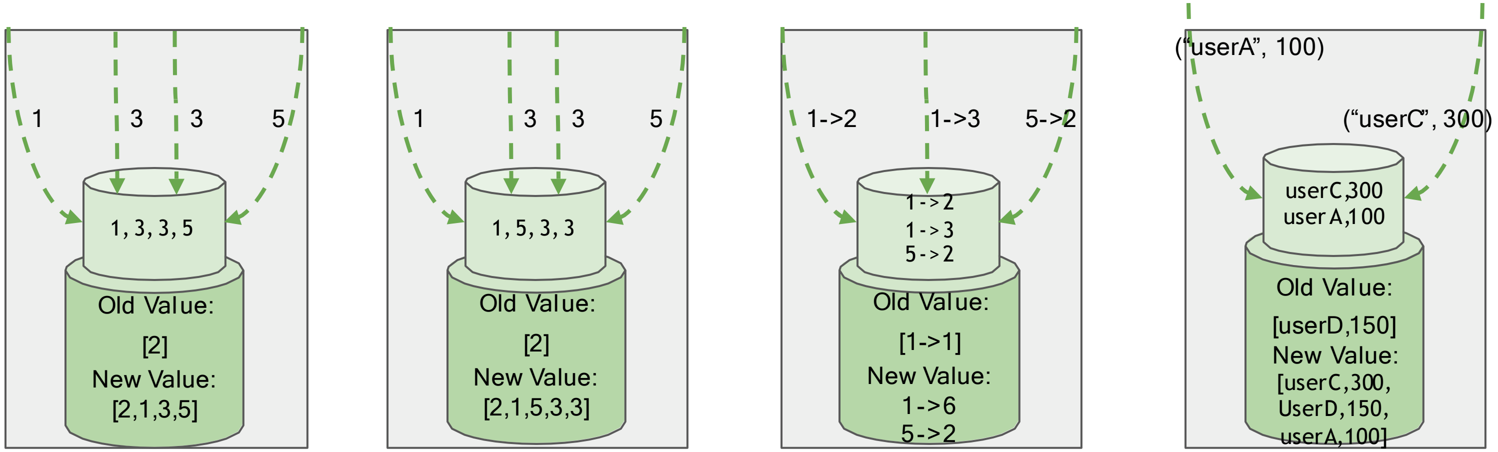


AvgAccum

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

Accumulators

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



SetAccum<int>

Maintains a collection of unique elements.

ListAccum<int>

Maintains a sequential collection of elements.

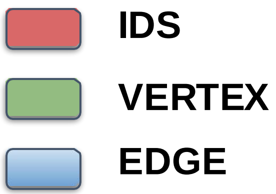
MapAccum<int,SumAccum<int>>

Maintains a collection of (key -> value) pairs.

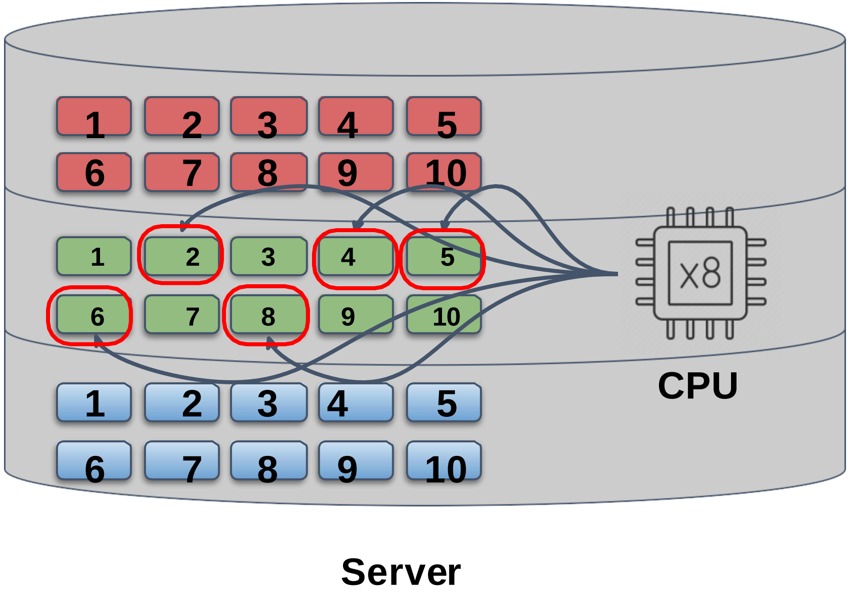
HeapAccum<Tuple>

Maintains a sorted collection of tuples and enforces a maximum number of tuples in the collection

MPP mechanism of TigerGraph



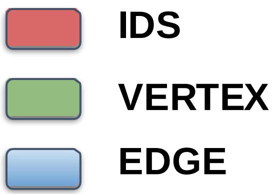
Processing Vertex-Induced ACCUM/WHERE clause or POST-ACCUM/HAVING clause



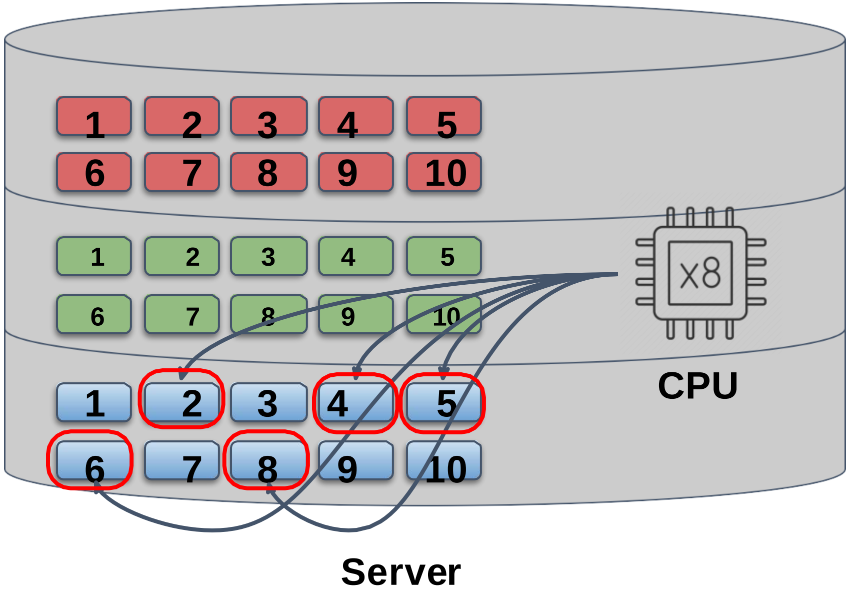
user_set = {User.*};
user_set = SELECT s FROM user_set:s
POST-ACCUM
…. // some logic;

- A thread will be assigned to each vertex segment to perform the logic defined in the **POST-ACCUM** clause in parallel.
- Once the task of one segment is done, the thread move to next unprocessed segment.
- By default, the maximum # of CPU cores of a thread will be assigned.

MPP mechanism of TigerGraph



Processing Edge-Induced WHERE/ACCUM clause

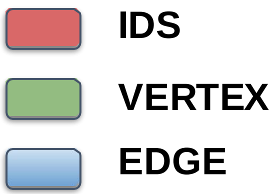


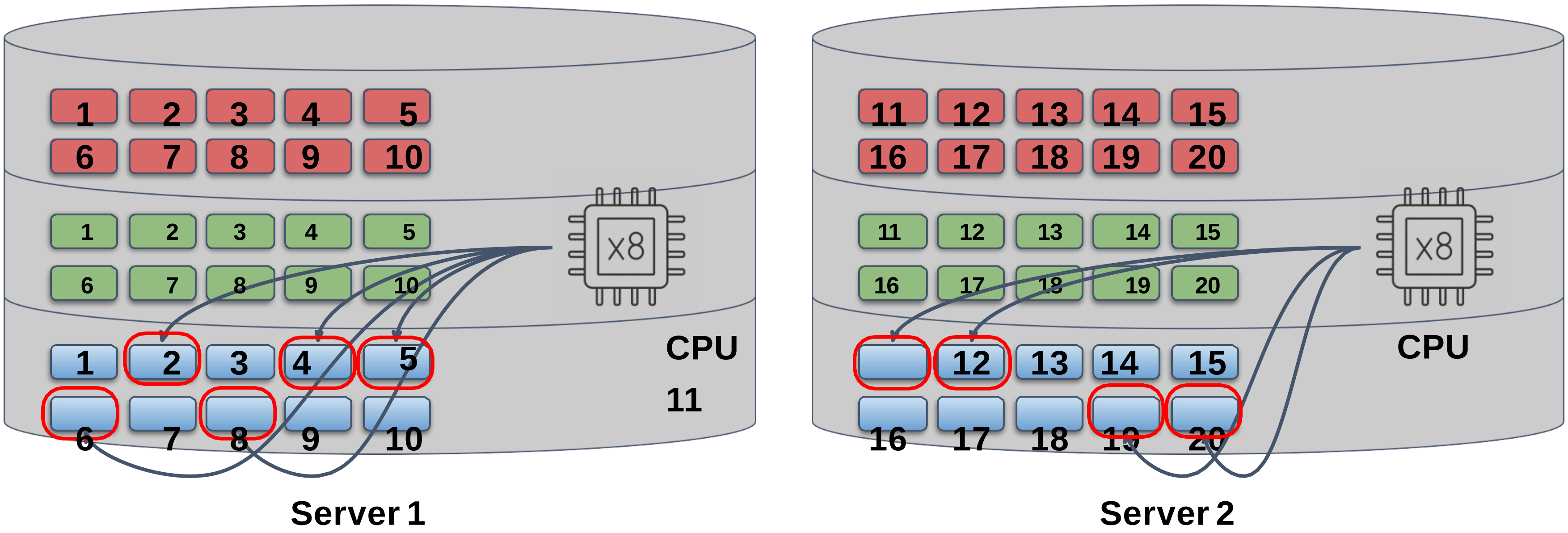

```
user_set = {User.*};
user_set = SELECT s FROM user_set:s-(:e)->:t
           ACCUM
           …. // some logic;
```

- A thread will be assigned to each edge segment to perform the logic defined in **ACCUM** clause in parallel.
- Once the task of a segment is done, the thread move to next unprocessed segment.
- By default, the maximum # of CPU cores of a thread will be assigned

MPP mechanism of TigerGraph

Processing Edge-Induced WHERE/ACCUM clause distributedly





Algorithm Types

- Centrality
  Assign numbers or rankings to each vertex corresponding to their network position
- Classification
  Classify the vertices into sets according to some external rule
- Community
  Group the vertices so that each group is densely connected
- GraphML/Embeddings
  Convert the neighborhood topology of each vertex into a fixed size vector of decimal values
- Path
  Find the best paths from one vertex to another (shortest, lowest weight, or other criteria)
- Similarity
  Compute similarity between pairs of items
- Topological Link Prediction
  Predict the existence of a link between two entities in a network
- Frequent Pattern Mining
  Find subgraph patterns that occur the most frequently

Algorithms

- Centrality
  - PageRank
  - Article Rank
  - Betweenness
  - Closeness
  - Degree
  - Eigenvector
  - Harmonic
  - Influence Maximization
- Community
  - Connected Components
  - K Core
  - K Means
  - Label Propagation
  - Local Cluster Coefficient
  - Louvain
  - Speaker-Listener Label Propagation
  - Triangle Counting
- GraphML/Embeddings
  - FastRP
  - Node2Vec
- Path
  - Astar_shortest_path
  - BFS
  - Cycle_detection
  - Estimated_diameter
  - Maxflow
  - Minimum_spanning_forest
  - Minimum_spanning_tree
  - Shortest Path
- Classification
  - Greedy Graph Coloring
  - Maximal_independent_set
- Similarity
  - Cosine
  - Jaccard
  - K Nearest Neighbors
  - Approximate Nearest Neighbors
- Topological Link Prediction
  - Adamic Adar
  - Common Neighbors
  - Preferential Attachment
  - Resource Allocation
  - Same Community
  - Total Neighbors

최단 경로 및 페이지 순위와 같은 일부 항목은 여러 가지 변형이 있으므로 총 개수가 50개가 넘습니다.

https://github.com/tigergraph/gsql-graph-algorithms

https://github.com/tigergraph/graph-ml-notebooks