Blazing fast distributed graph query processing: 100x faster than Spark

Sungpack Hong, Research Director
Vasileios Trigonakis, Principal Member of Technical Staff
Tomas Faltin, Research Assistant

Oracle Labs
2019/9/17

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

Abstract

- A key challenge in graph querying and pattern matching is processing increasingly large graphs and queries that don't fit in a single machine's memory. This session presents Oracle Labs' novel technology PGX.D/Async, which can be used with Oracle Big Data Spatial and Graph to perform blazing-fast scalable distributed pattern matching for property graphs. In the session's first part, developers will learn PGQL, a SQL-like graph query language, and the secret behind how top performance is achieved with asynchronous almost-depth-first traversal, allowing for high parallelism and precise control over memory consumption. The second part illustrates a performance analysis showing how PGX.D/Async performs 100x as fast as Spark and handles cases that Spark cannot.

*Please note that this session discusses research and development efforts, not shipping product features. Oracle's Safe Harbor provisions apply to the contents of this presentation.

Contents

Graph Data Model and Graph Query

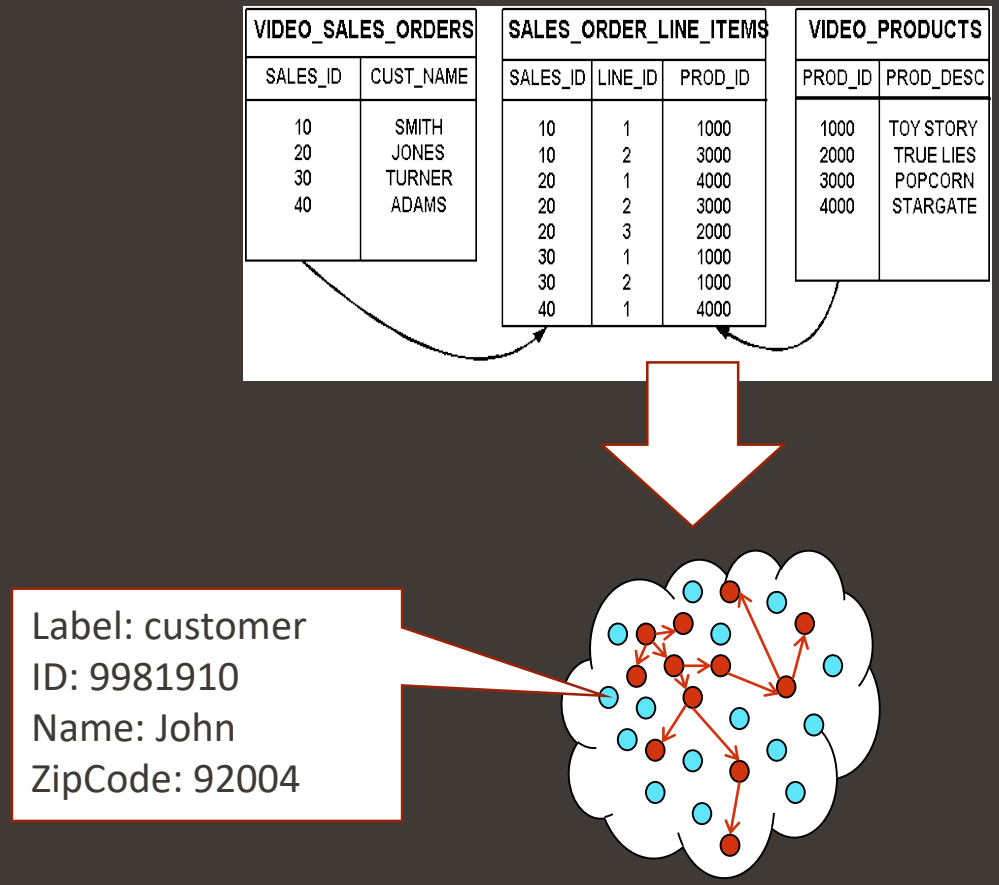
Distributed Graph Processing | Research & Development

Asynchronous Traversal on Distributed Graphs

Performance Comparison

Graph Data Model

- Representing your data as Graph
  - Data entities as vertices
  - Relationships as edges
  - Vertices/Edges can have attributes, or properties
- Graph model explicitly captures relationship between data entities

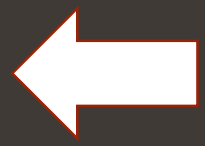


Graph Data Model: Benefits

Powerful data analytics for business insight

Visualization and interactive data browsing

Easy query and fast navigation of complex data

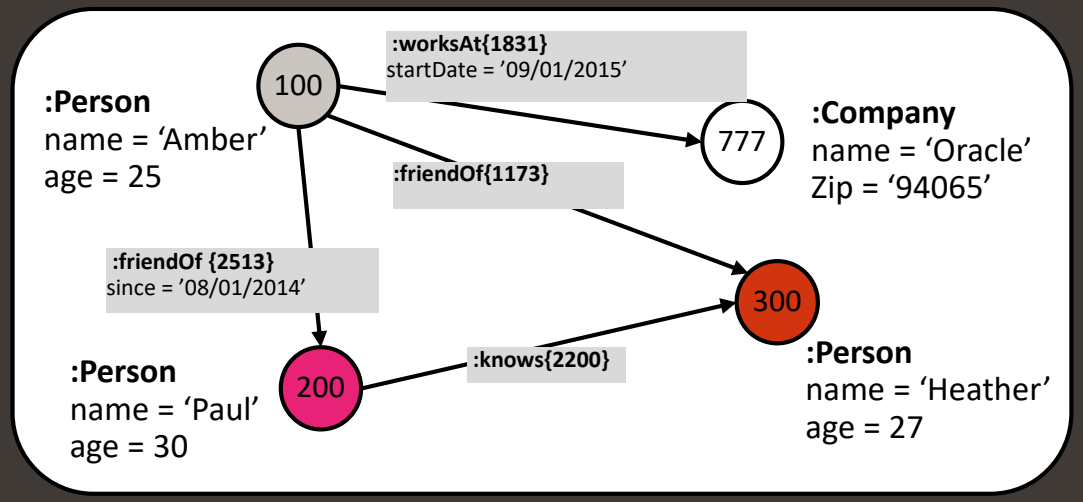


Come to other graph talks

at OOW and Code One !

Graph Query: Example

Find all instances of a given pattern in the data graph



Query: Find all people who are known to friends of 'Amber'.

```
SELECT v3.name, v3.age
  FROM myGraph
 MATCH (v1:Person) -[:friendOf]-> (v2:Person) -[:knows]-> (v3:Person)
 WHERE v1.name = 'Amber'
```

PGQL – Property Graph Query Language

Familiar SQL-like syntax

- SELECT .. FROM .. WHERE ..
- GROUP BY, HAVING, ORDER BY, etc.

Graph pattern matching

- Define a high-level pattern and match all the instances in the data graph

Recursive path expressions

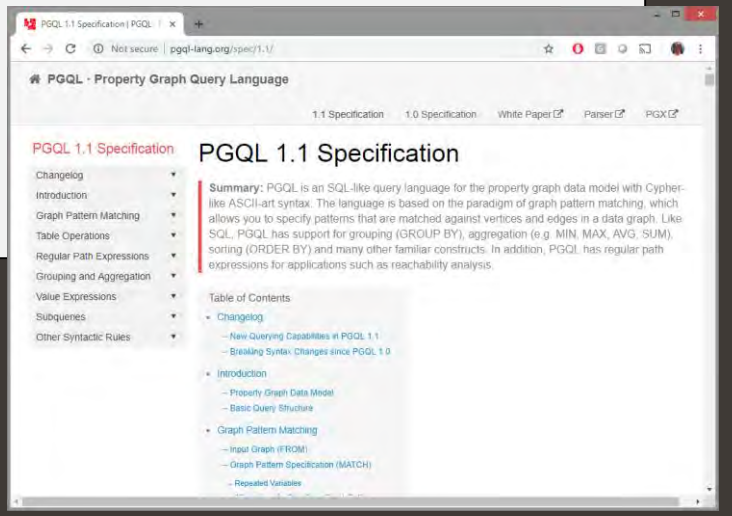
- E.g. can I reach from vertex A to vertex B via any number of edges?

PGQL has been input to ongoing standardization efforts

- i.e. SQL extension for property graph

Example query:

```
SELECT p2.name AS friend_of_friend

FROM friendGraph
MATCH (p1:Person) -/:friend_of{2}/->
                                  (p2:Person)

GROUP BY ..

ORDER BY ..

LIMIT ..

OFFSET ..
```



http://pgql-lang.org/

PGQL compared to SQL

Graph query is intuitive and succinct

Query: Find all the devices that are (transitively) connected to HVMV8

```
PATH connects_to AS (from) <- (connector) -> (to)
SELECT y.name
 MATCH (x:Device) -/:connects_to*/-> (y:Device)
 WHERE x.name = 'Regulator, HVMV_8'
   AND x <> y
```

PGQL

```
WITH temp(device_id, device_name) AS (
  -- Anchor member:
  SELECT device_id, name
  FROM   Devices
  WHERE  name = 'Regulator, HVMV_B‘
UNION ALL
  -- Recursive member:
  SELECT Devices.device_id, Devices.name
  FROM   temp, Devices, Connections conn1,
         Connections conn2, Connectors
  WHERE  temp.device_id = conn1.to_device_id
    AND  conn1.from_connector_id = Connectors.connector_id
    AND  Connectors.connector_id = conn2.from_connector_id
    AND  conn2.to_device_id = Devices.device_id
    AND  temp.device_id != Devices.device_id)
CYCLE device_id SET cycle TO 1 DEFAULT 0
SELECT DISTINCT device_name
FROM temp
WHERE cycle = 0
  AND device_name <> 'Regulator, HVMV_B'
```

SQL

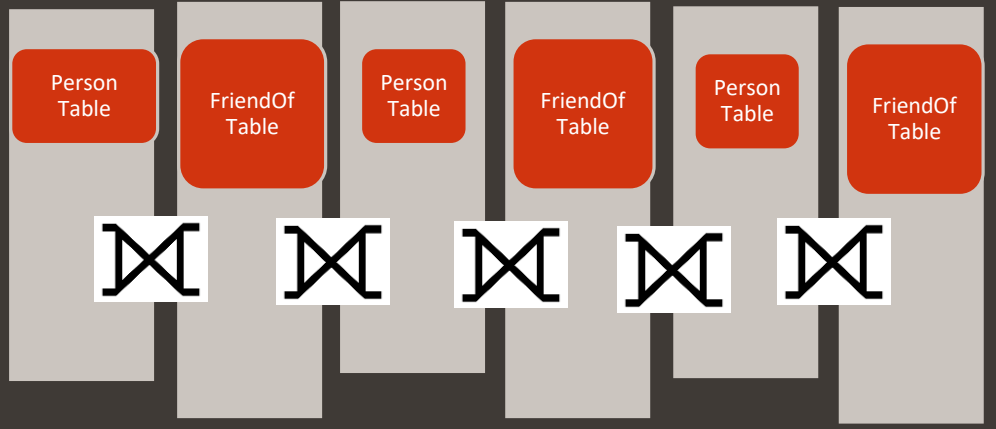
Execution of Graph Query

- Since Graph query is functionally equivalent to SQL, we can execute graph query by translating it into SQL

  ➔ Each edge traversal corresponds to joining of edge and vertex table(s)

  ➔ Can end up with a lot of joins

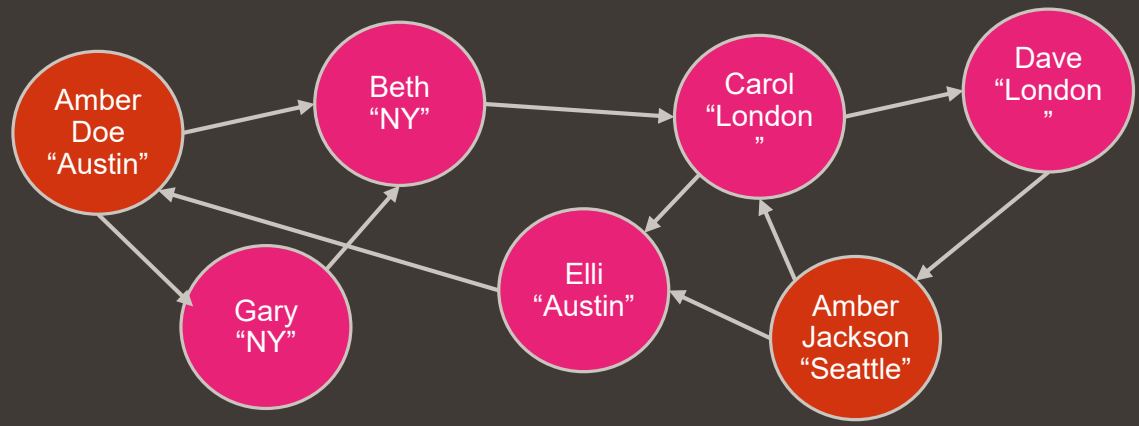
```
SELECT v3.name, v3.age
  FROM myGraph
 MATCH (v1:Person) -[:friendOf]-> (v2:Person) -
[:knows]-> (v3:Person)
 WHERE v1.name = 'Amber' and (v3.city = v2.city)
```



Execution of Graph Query

- Alternative is using graph index
  - ➔ Vertices has list of edges
  - ➔ An edge directly points to corresponding neighbor vertex
  - ➔ Allows to iterate graph
- Graph index typically builds in-memory

```
SELECT v3.name, v3.age
  FROM myGraph
 MATCH (v1:Person) -[:friendOf]-> (v2:Person) -
[:knows]-> (v3:Person)
 WHERE v1.name = 'Amber' and (v3.city = v2.city)
```



Research Area

Distributed Graph Processing

System and Bulk-Synchronous Execution for Algorithm [SC 2015]

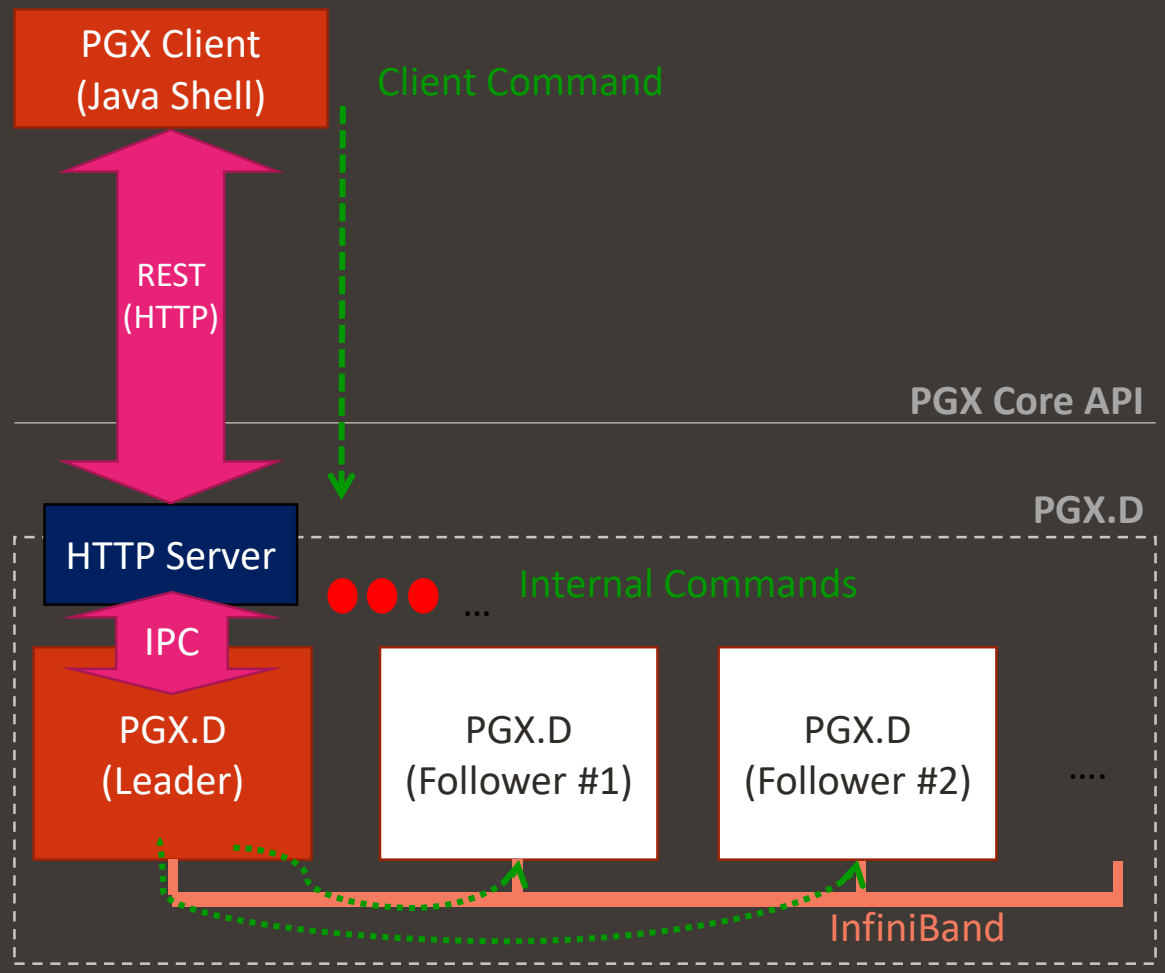
Distributed Execution For Large Graphs

PGX.D: Distributed Runtime of PGX

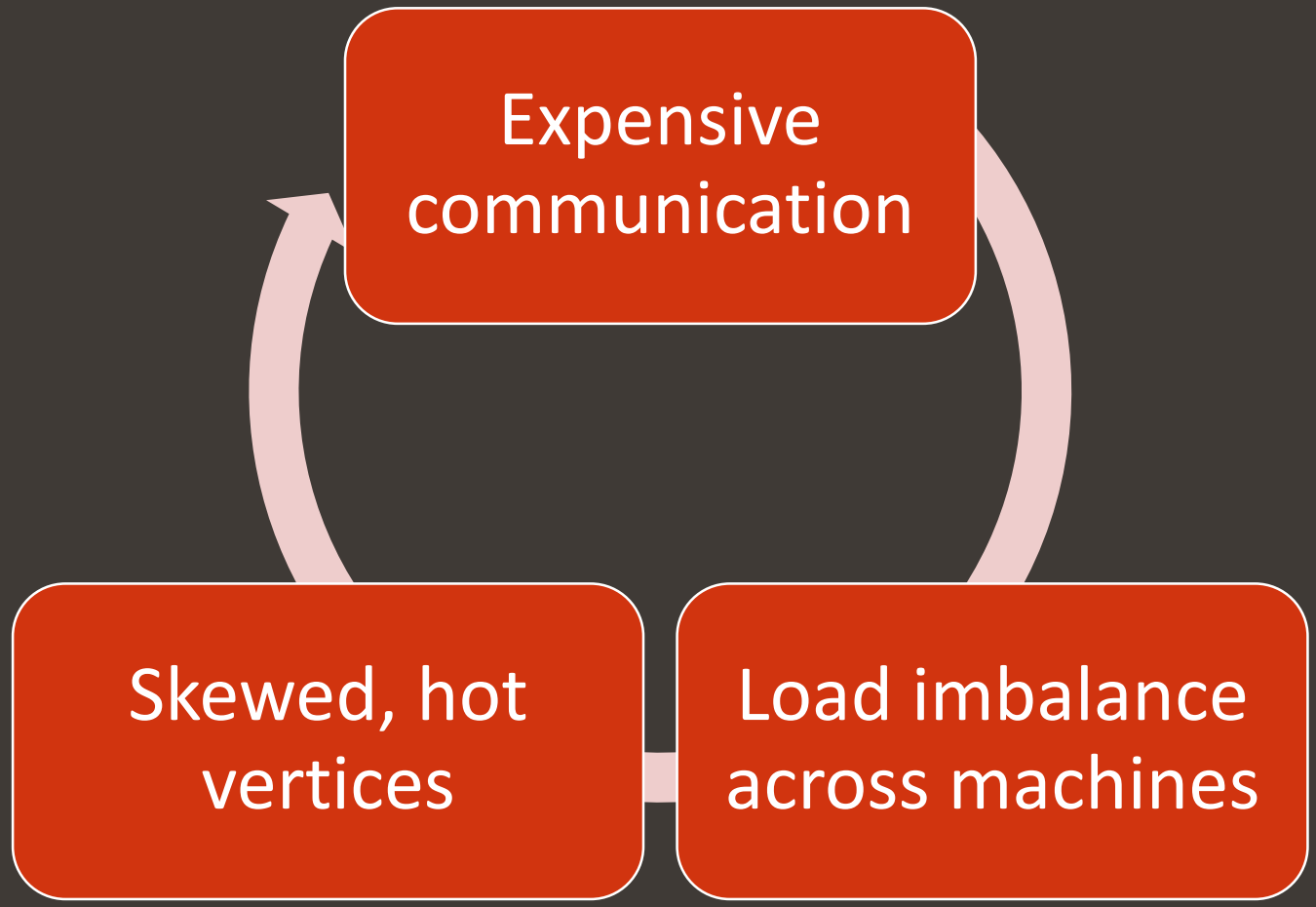
- Capable of loading large graphs in-memory
- The graph is partitioned and loaded into multiple machines
- Graph algorithms are run in a distributed way using remote communication

Goal to support the same API as PGX

- Currently supports a subset
- A PGX Client can connect to PGX.D in the same way as to a PGX.SM server

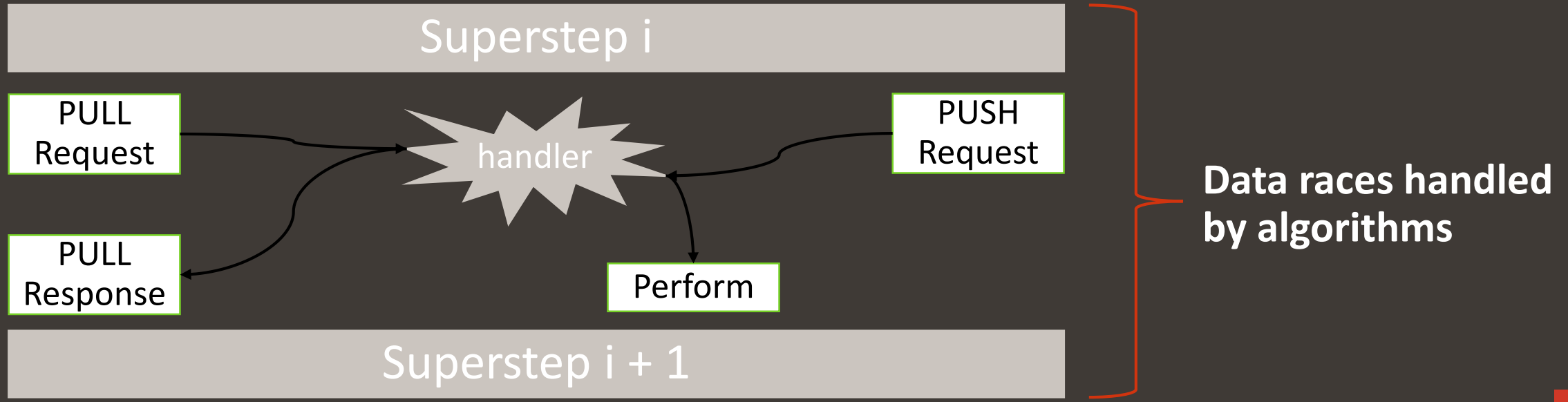


Key Challenges For Distributed Graph Algorithms



Problem: Expensive Communication
Solution: PGX.D Execution Model

- Relaxed bulk-synchronous execution
  - Allows one-sided reading of remote data in a single iteration
  - Applies local and remote write requests immediately
- Supports both PUSH and PULL patterns



Problem: Load Imbalance Across Machines
Solution: Partitioning with Edge-Chunking

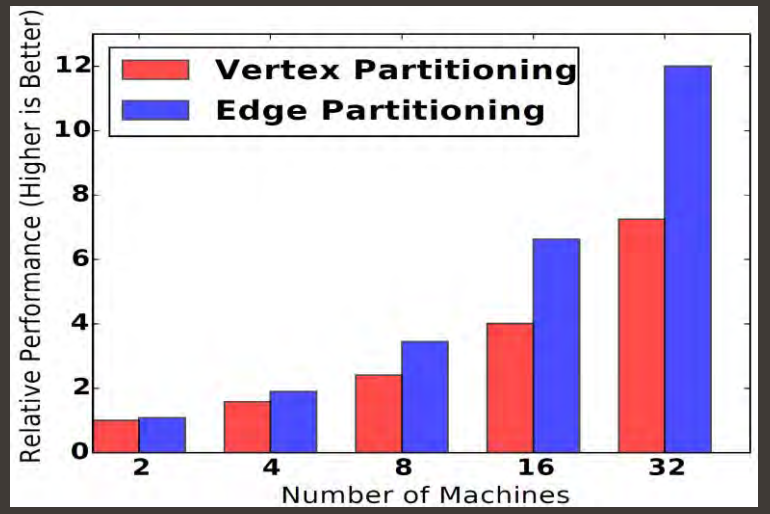
Vertex chunking

~Same # of **vertices** per machine

Problem: Iteration imbalance
Machine 0 has more edges =
more work than Machine 1

Edge chunking

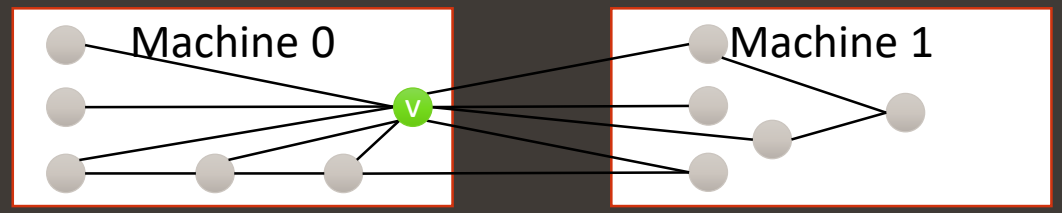
~Same # of **edges** per machine



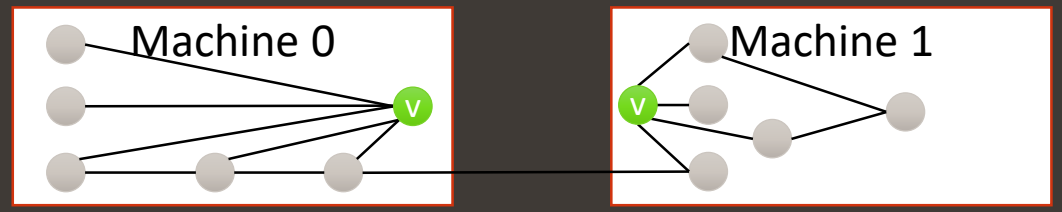
Twitter graph Pagerank

Problem: Skewed, Hot Vertices
Solution: Replication of High-Degree Vertices

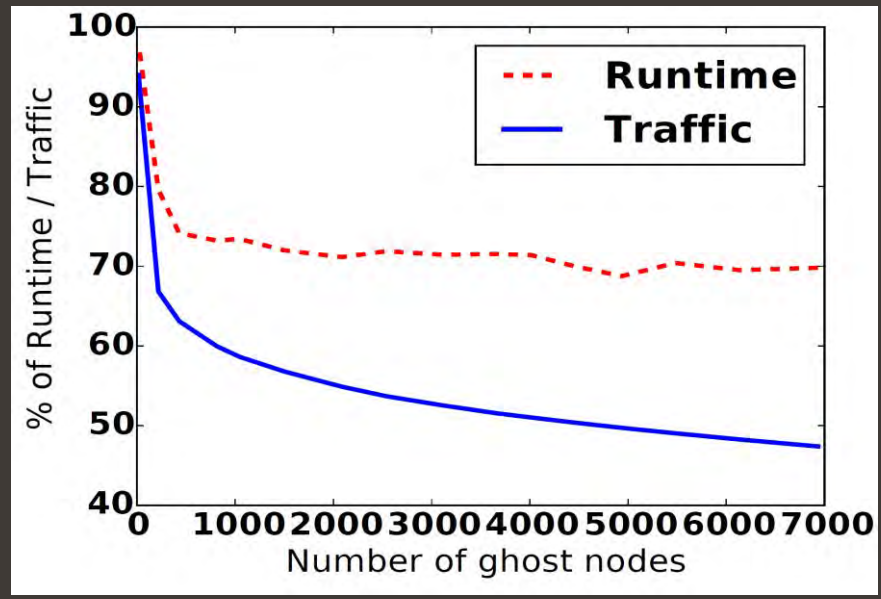
- High-degree vertices cause imbalance and remote traffic



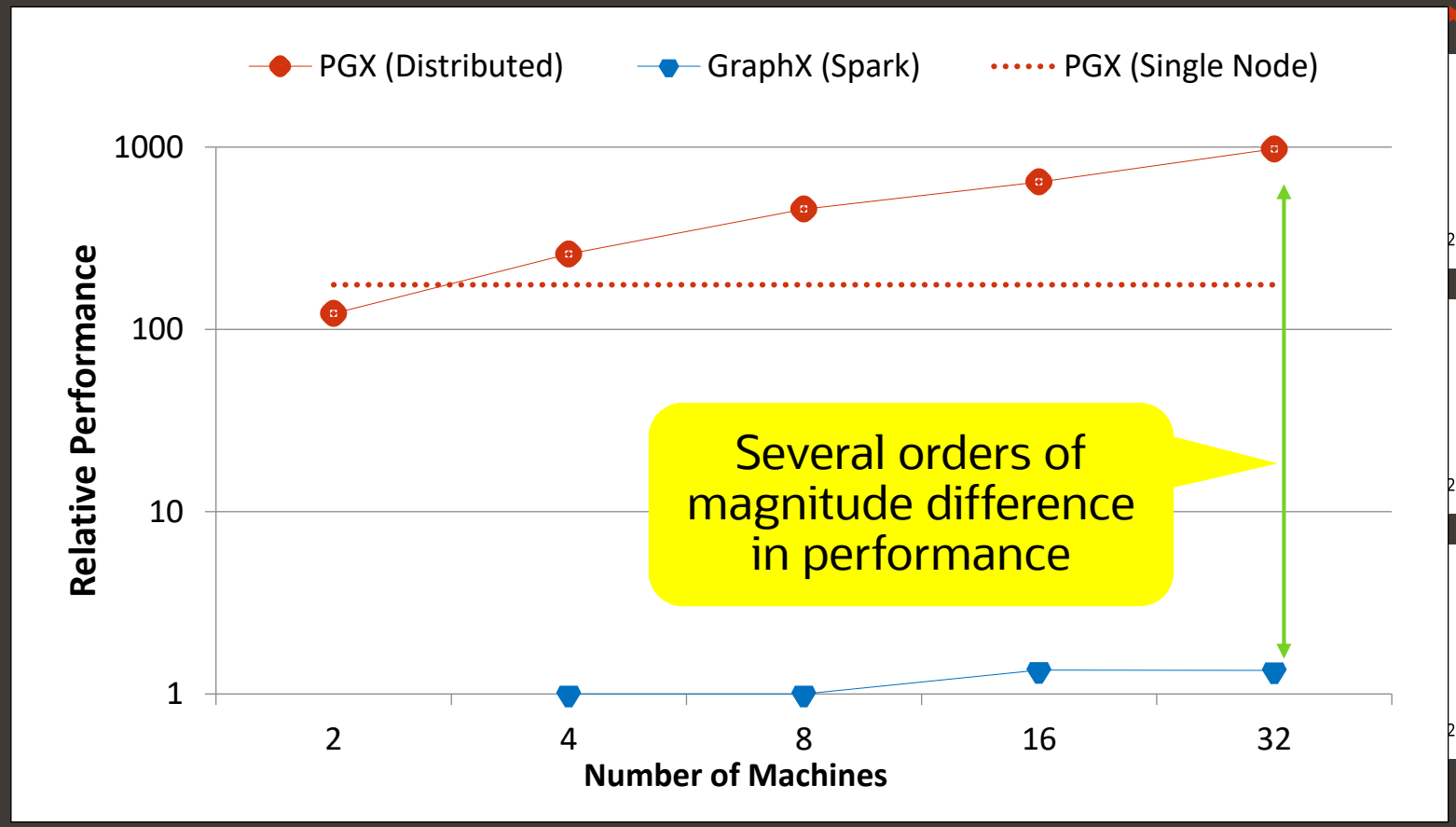
- Ghost vertices
  - Replicate high-degree vertices, split edges
  - Synchronize their properties between supersteps

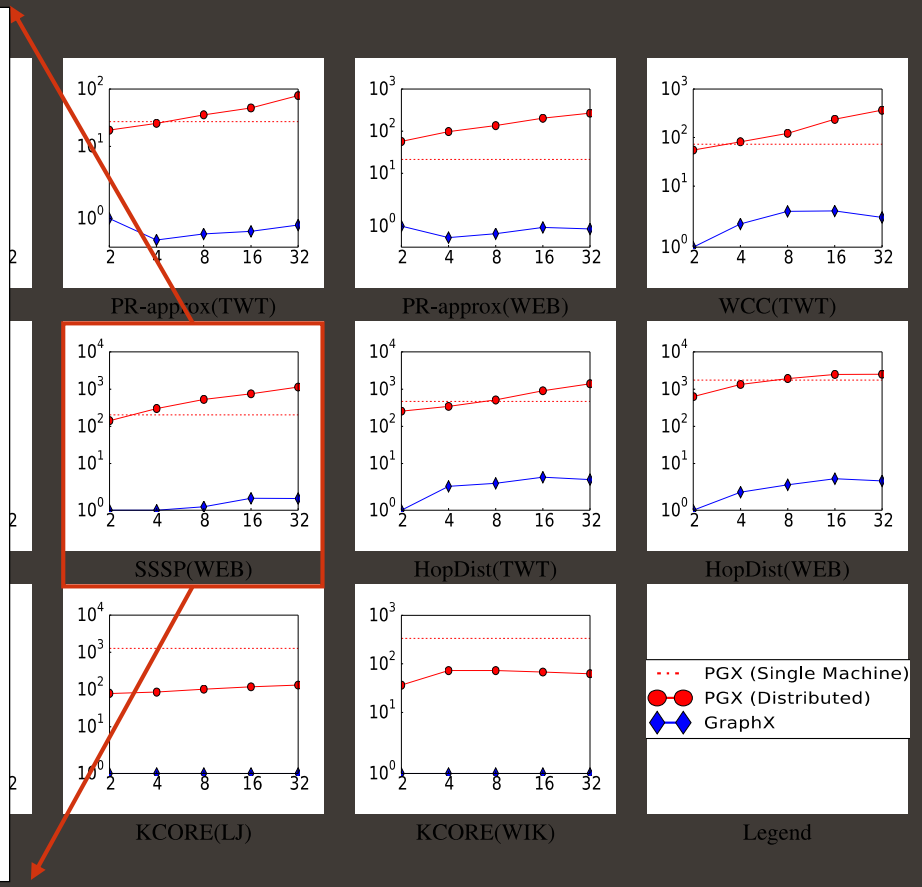


Twitter graph, Pagerank



Algorithm: PGX.D vs. Apache Spark (GraphX)






```
Hardware:       Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz - 256 RAM
Network:        Infiniband (40Gbps)
```

Algorithm: PGX.D Weak-Scaling

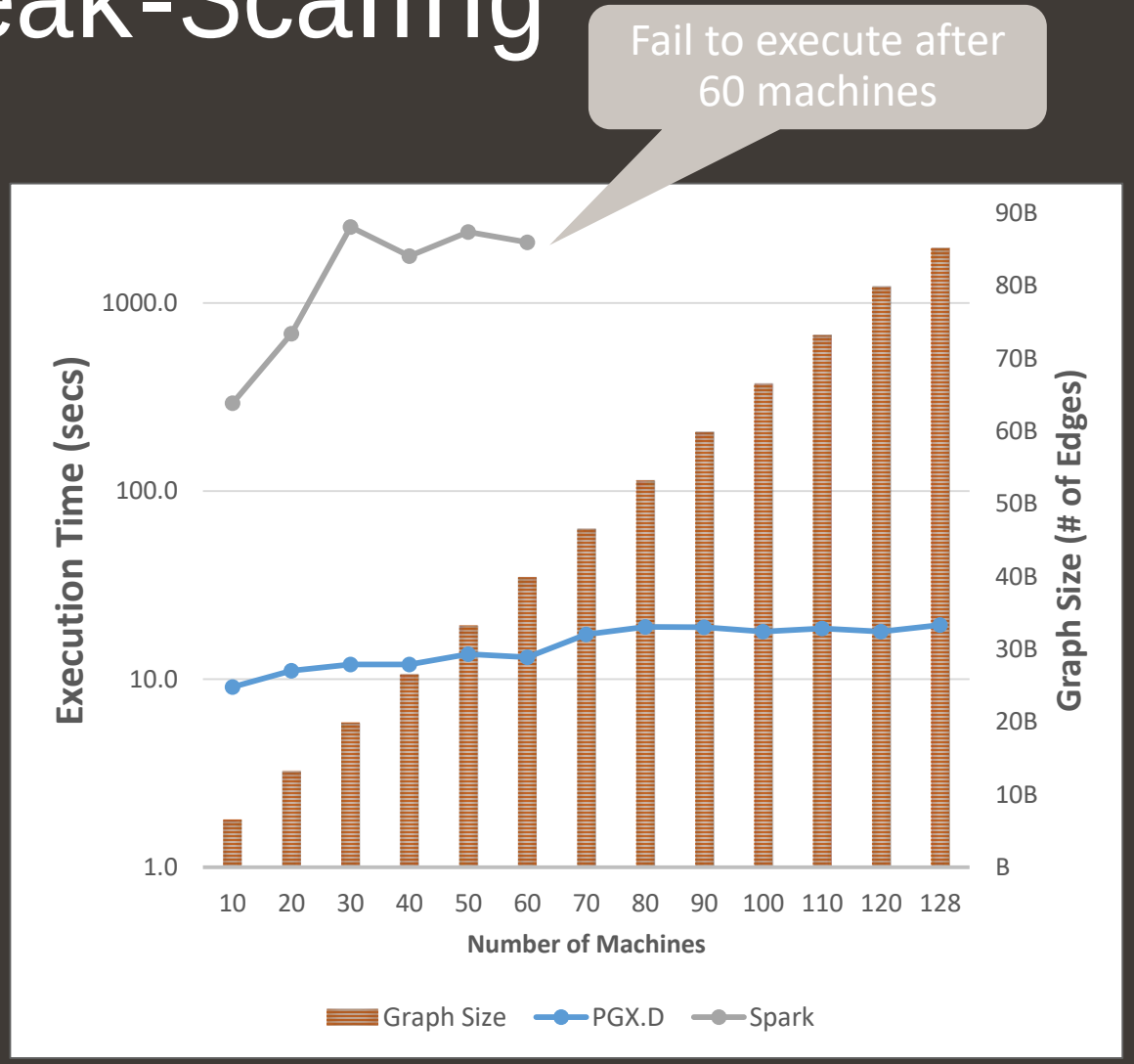
Experiments

- Increasing number of machines with proportionally increasing graph size (number of edges)
- Measure execution time of algorithm (pagerank_exact)
- Show the execution time does not grow

Comparisons

- Apache Spark (Graph X)

- Machines: X5-2 or similar grade
- Network: Infiniband 54 Gbps
- Graph Data: Generated by BTERonH (UPC ERO)



Algorithm: PGX.D Strong-Scaling

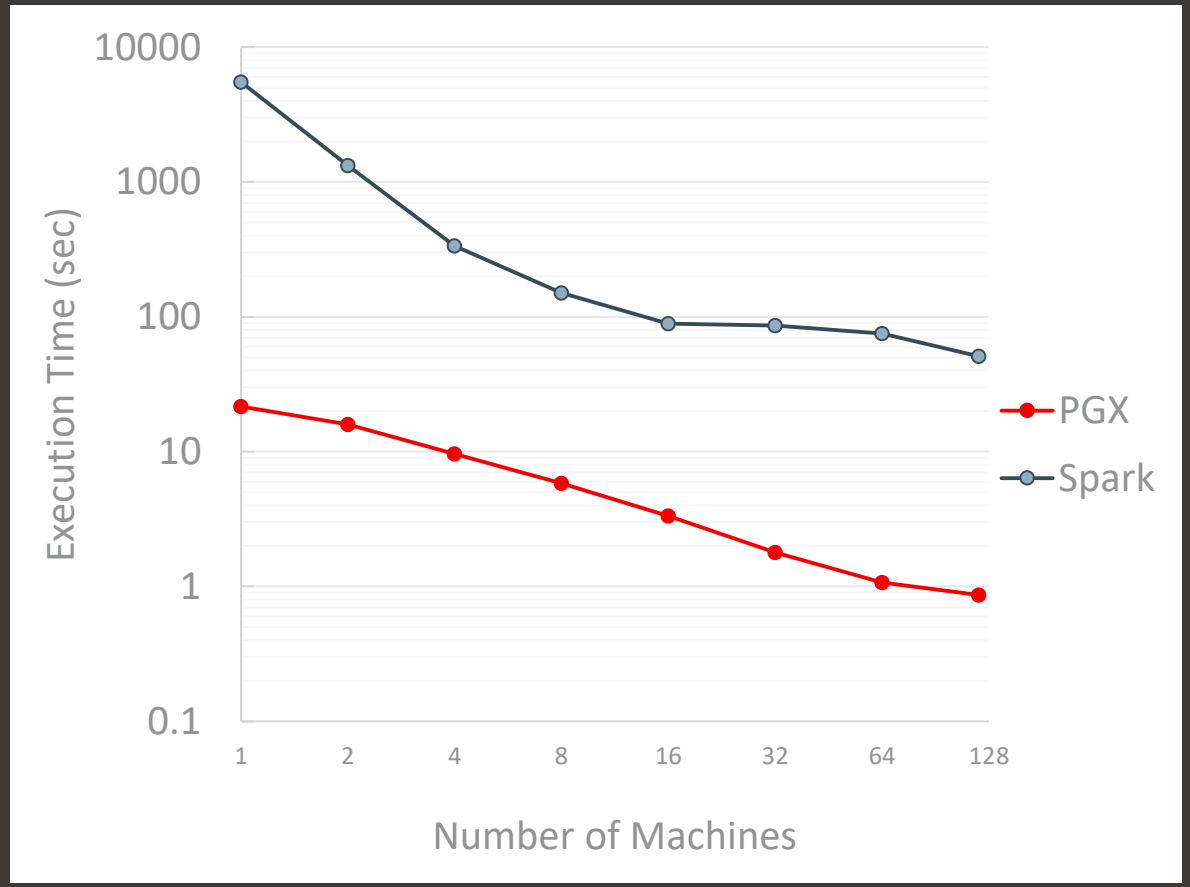
Experiments

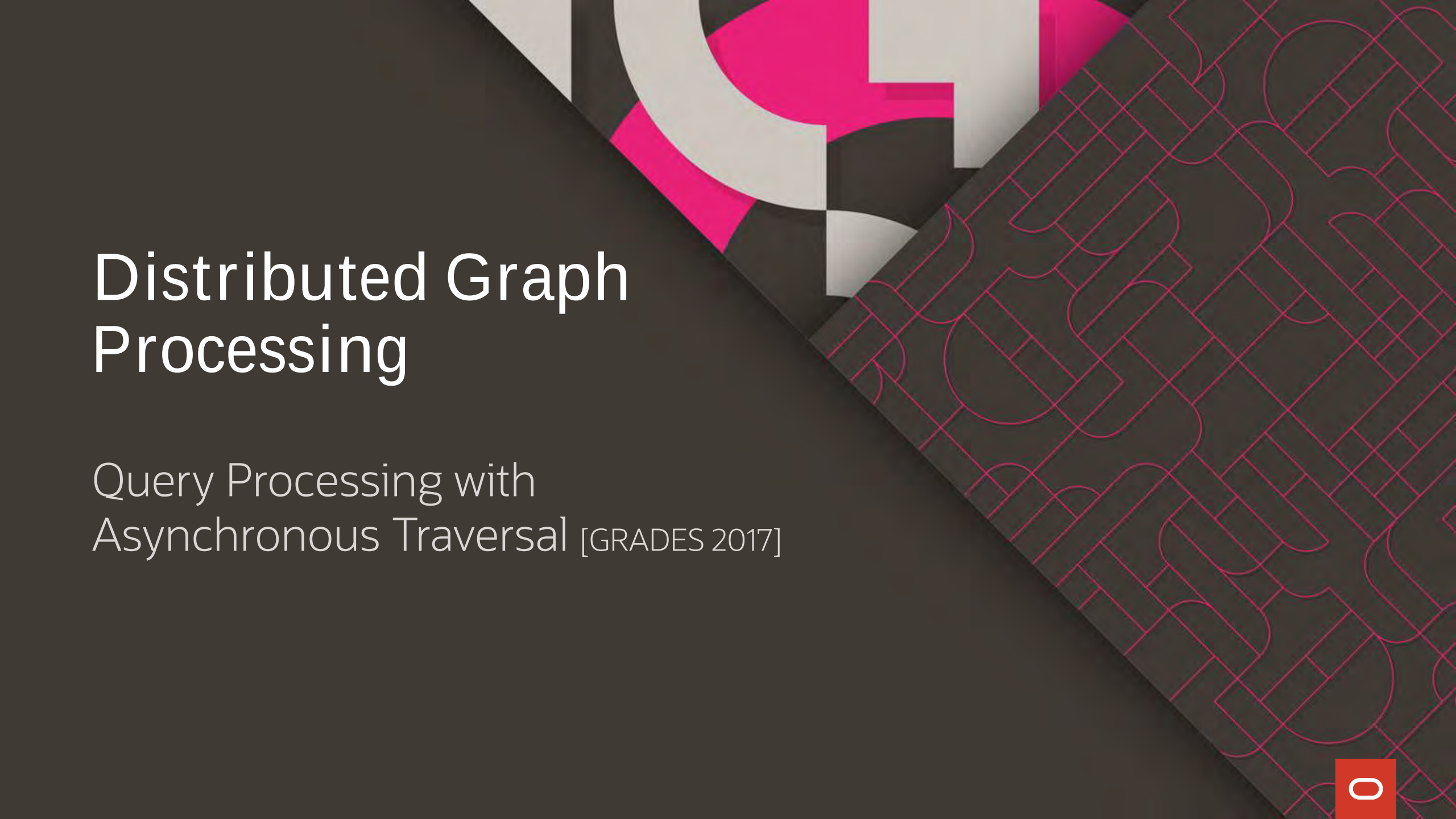
- Increasing number of machines with fixed graph
- Measure execution time of algorithm (pagerank_exact)

Comparisons

- Apache Spark (Graph X)

- Machines: X5-2 or similar grade
- Network: Infiniband 54 Gbps
- Graph Data: Twitter Graph – 1.4 Billion Edges





Distributed Graph Processing

Query Processing with Asynchronous Traversal [GRADES 2017]

Executing Distributed Graph Queries

- How to do this? What are the options?
- (Option 1) Translate query to SQL

  We know how to do distributed joins
- (Option 2) Use bulk-synchronous graph engine

  We already have this one

  Emulate graph traversing with bulk-synch computation
- .. Both options come with some challenges

Explosion of Partial Solutions

Twitter graph

Number of (partial) solutions can grow very large, after a few hops of matching

➔ Problem for both Distributed Join and Bulk-synch engine

➔ Cannot hold all (partial) solutions in-memory

```
SELECT COUNT(*) MATCH (a)
+----------------------+
|    COUNT(*)          |
+----------------------+
| 41,652,230           |
+----------------------+
```

0 hops

```
SELECT COUNT(*) MATCH (a)->()
+----------------------+
|    COUNT(*)          |
+----------------------+
| 1,468,365,182        |
+----------------------+
```

1 hop

```
SELECT COUNT(*) MATCH (a)->()->()
+----------------------+
|    COUNT(*)          |
+----------------------+
| 9,324,563,362,739    |
+----------------------+
```

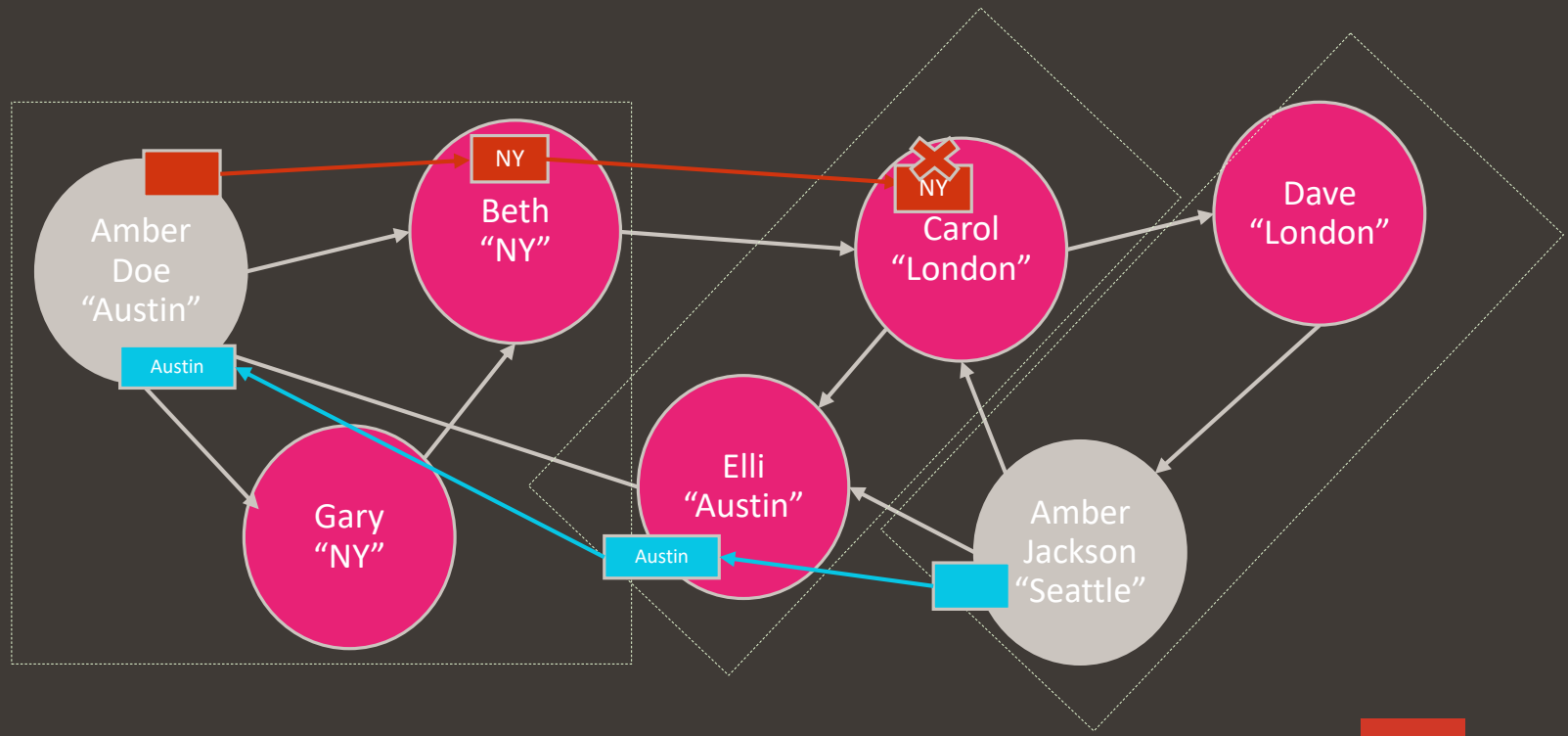
2 hops

Our Approach: Asynchronous Distributed Graph Traversal

- Implement specialized graph traversal on distributed setting
- Built on message passing
- Context is carried over during traversal

  (For projection + cross constrains)

```
SELECT v3.name, v3.age
  FROM myGraph
 MATCH (v1:Person) -[:friendOf]-> (v2:Person)
         -[:knows]-> (v3:Person)
 WHERE v1.name = 'Amber' and (v3.city = v2.city)
```



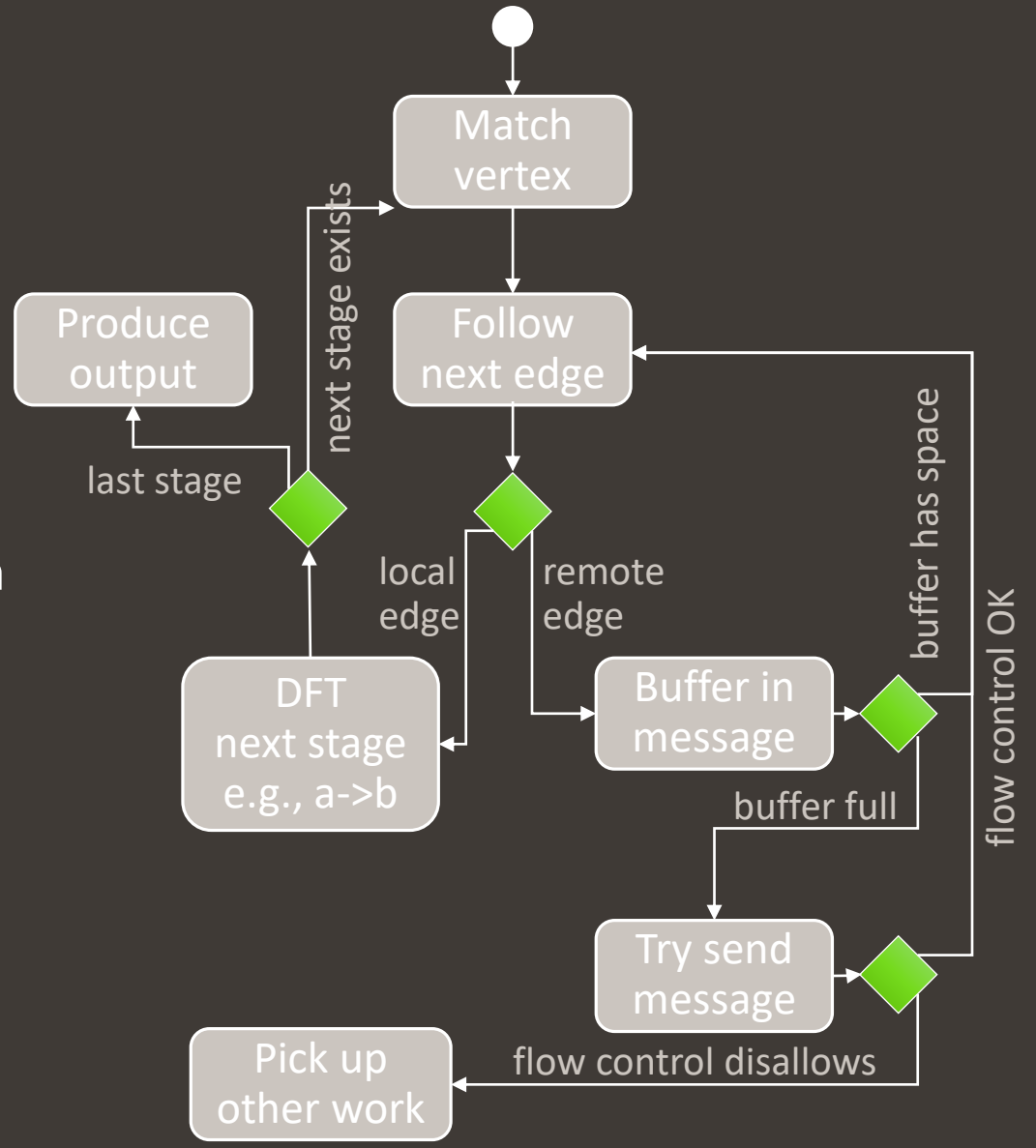
Traversal Engine Design

1. Asynchronous communication
   - No supersteps, no blocking
   - A matching step that requires remote data is sent to the remote machine; and the worker continues with other local work

   → Workers do not block due to remote communication

2. Depth-first traversal (DFT)
   - Eager fine-grained completion of matches
   - Allows for flow control

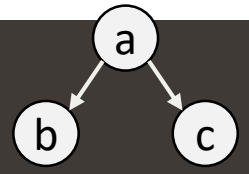
   → Control memory / network consumption



Query Planning: Logical and Physical

PGQL

```
SELECT a, b.name
MATCH
  (a)-[]->(b),
  (a)-[]->(c)
WHERE
  id(a) < 17,
  a.type = b.type,
  b.type != c.type
```



query plan

```
<root vertex match>
  (a: id < 17)
<neighbor match>
  (a->b: a.type = b.type)
<neighbor match>
  (a->c: b.type != c.type)
```

distributed query plan

```
<stage a: id < 17>
  hop: neighbor match
    (a->b: a.type = b.type)
<stage b>
  hop: inspection step
    (b~>a)
<stage a>
  hop: neighbor match
    (a->c: b.type != c.type)
<stage c>
  hop: output
```

distributed execution

```
Stage 0: (a)
  filter : id < 17
  hop    : out nghbr: b
  capture: a.type
  out ctx: +a
Stage 1: (b)
  filter : a.type = b.type
  hop    : inspection: a
  capture: b.type
  out ctx: +b.name
Stage 2: (a)
  hop    : out nghbr: c
Stage 3: (c)
  filter : b.type != c.type
  hop    : output
```

Design Outcome

1. Fully in-memory regardless of graph or query size
   Can push output to a storage medium and support terabyte-sized results
2. Controllable size of intermediate results
   Control memory consumption, network traffic, machine load
3. Guaranteed query termination
   Via incremental termination protocol
4. Solution modifiers (GROUP BY, ORDER BY) ➔ Table operators

Incremental Termination Protocol

Tracking per-machine completion of each stage

Send COMPLETED messages to detect termination

Termination condition: machine M, stage k

1. Have received COMPLETED messages for stage k-1 from all other machines
2. Have processed all received messages for stage k

→ Send COMPLETED to other machines

No more incoming messages for this stage can arrive

No more work for that stage (special case: stage 0)

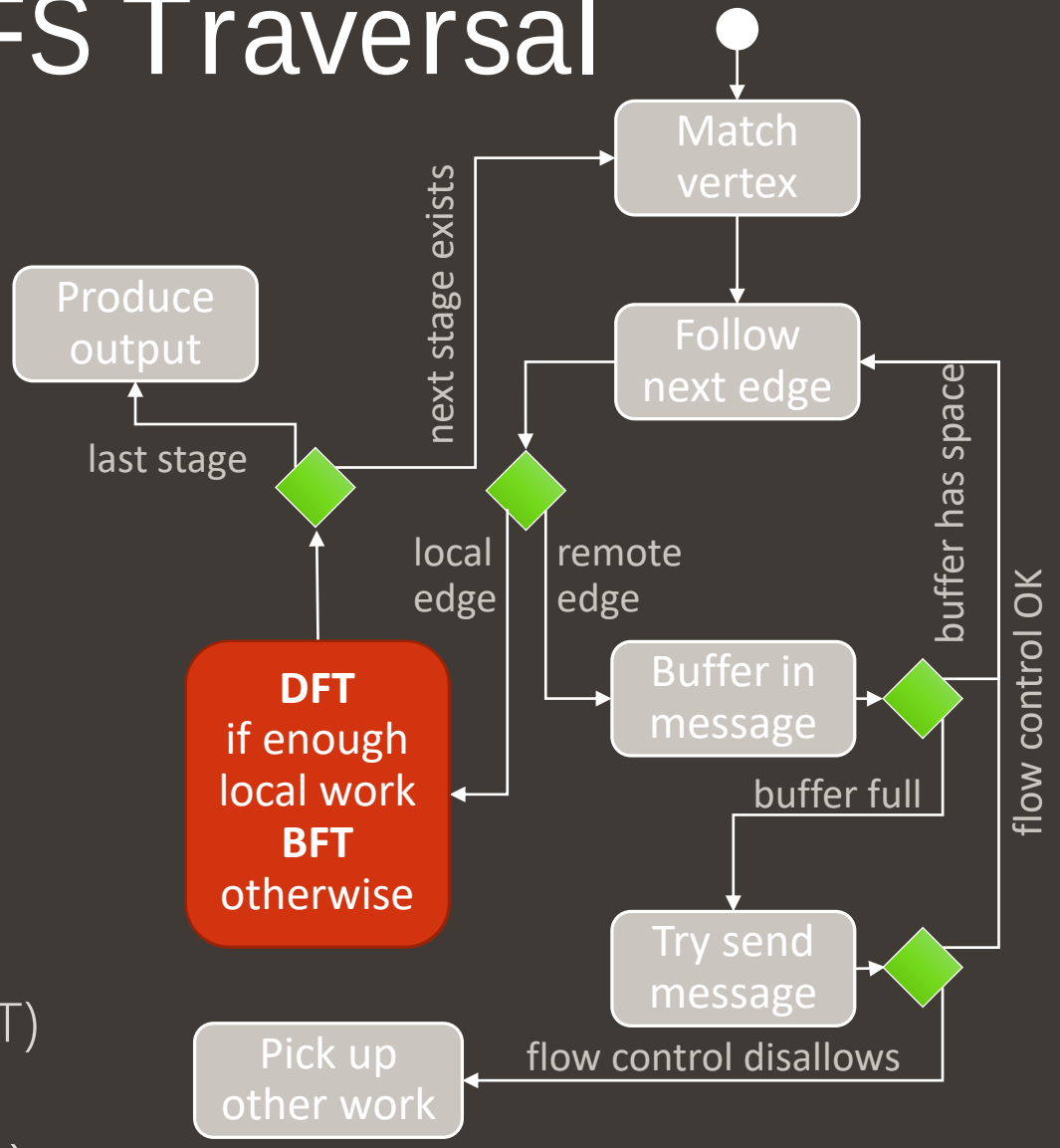
Optimization: Almost DFS Traversal

- DFT does not parallelize well on queries with “narrow” start

→E.g. LDBC Q23: 1 country, 10 cities, 3351 persons

```
SELECT COUNT(msg) AS messageCount, …
MATCH
  (person:person)<-(msg:post|comment)->(dst:country),
  (person)->(city:city)-[:isPartOf]->(homeCountry:country)
WHERE
  homeCountry.name = 'Egypt' AND homeCountry <> dst
```

- Combine DFS with BFS traversal
  - Take the best of both worlds
    - → Control memory/network consumption (DFT)
    - → Improve performance/parallelization (BFT)
  - (Asynchronous communication remains)



Performance Comparison

Query Processing with Asynchronous Traversal

LDBC SNB Benchmark

- Standardized benchmark with social network workloads
- Synthetic datasets of different scales
- Distributions and correlations similar to real social networks

- Business intelligence (BI) queries
  - The most complex queries testing larger part of the graph
  - Require extensive optimization across all the layers of the system (not only graph traversal)
  - We re-wrote some queries for our experiments
    - Sub-queries, regular path queries and having clause

LDBC – Example of BI Queries

Q8

```
SELECT relatedTag.name, COUNT(DISTINCT comment) AS count
MATCH (m:post|comment)-[:hasTag]->(tag:tag),
      (m:post|comment)<-[:replyOf]-(comment:comment)-[:hasTag]->(relatedTag:tag)
WHERE tag.name = 'Genghis_Khan'
GROUP BY relatedTag.name
ORDER BY count DESC, relatedTag.name
```

Q15

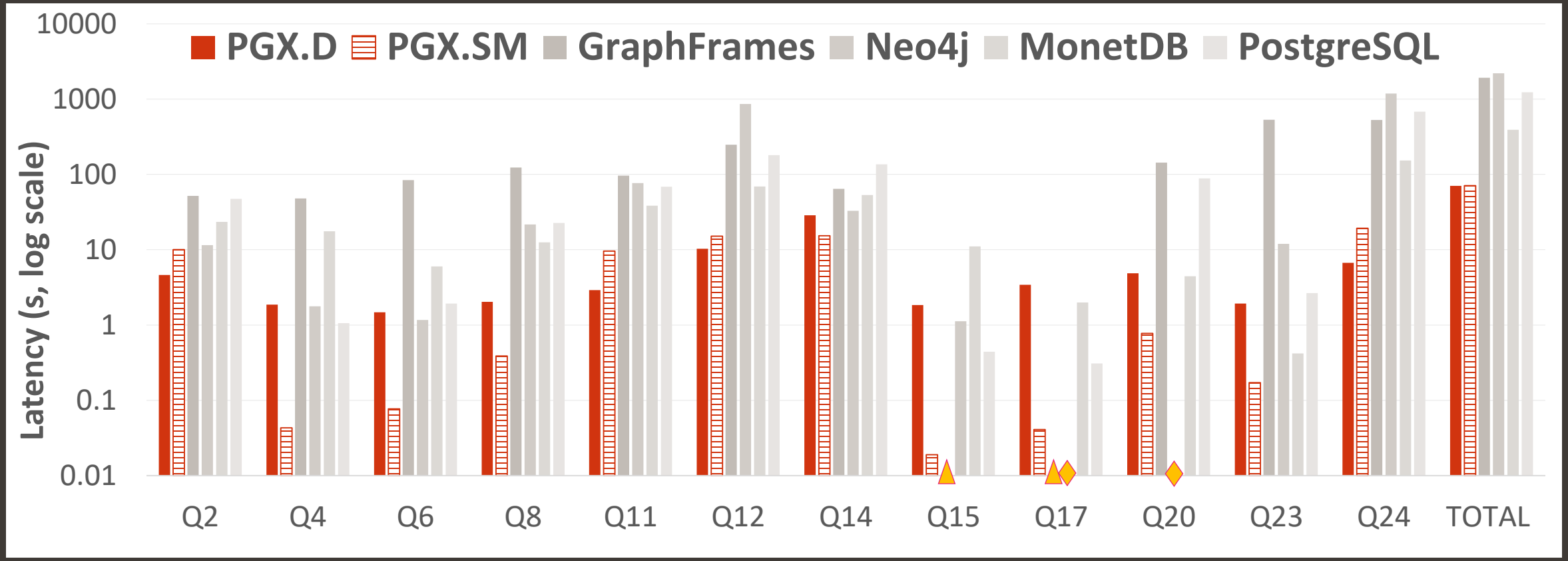
```
SELECT id(person), COUNT(*) AS count
MATCH (country:country)<-[:isPartOf]-(:city)<-[:isLocatedIn]-(person:person),
      (country)<-[:isPartOf]-(:city)<-[:isLocatedIn]-(friend:person),
      (person)-[:knows]-(friend)
WHERE country.name = 'Burma'
GROUP BY person
ORDER BY id(person)
```

Comparison to Apache Spark

- GraphFrames: Graph Query from Apache Spark
  - Distributed in-memory data processing engine
  - Support graph pattern query (motif)
  - Translate graph pattern matching into relational operators (i.e. Joins)

Comparing PGX to Open-Source Engines

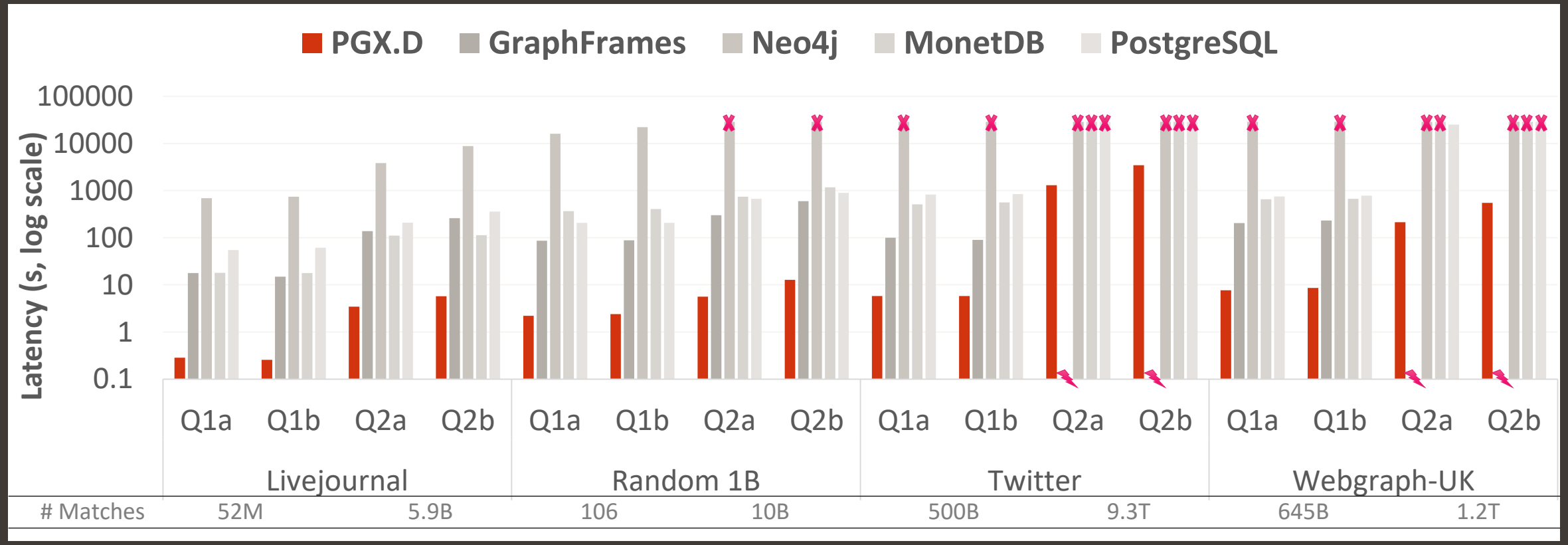
missing feature
Result mistmatch



LDBC 100 Social Graph (283M vertices, 1.78B edges) and Queries
PGX.D and GraphFrames on 8 machines

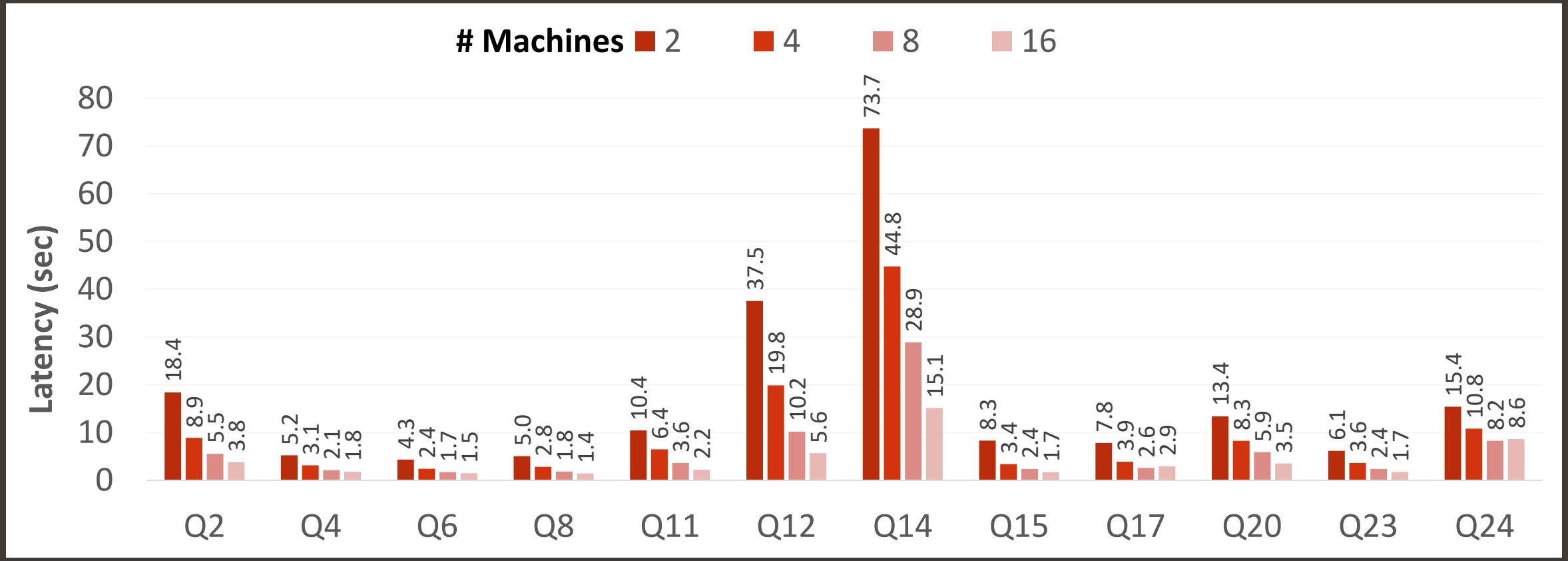
Comparing PGX to Open-Source Engines

cancel after 8 hours
hang, out of memory



Various graphs of increasing size. Q1: (v1)->(v2)->(v1), Q2: (v1)->(v2)->(v3)
PGX.D and GraphFrames with 8 machines with 756GB each

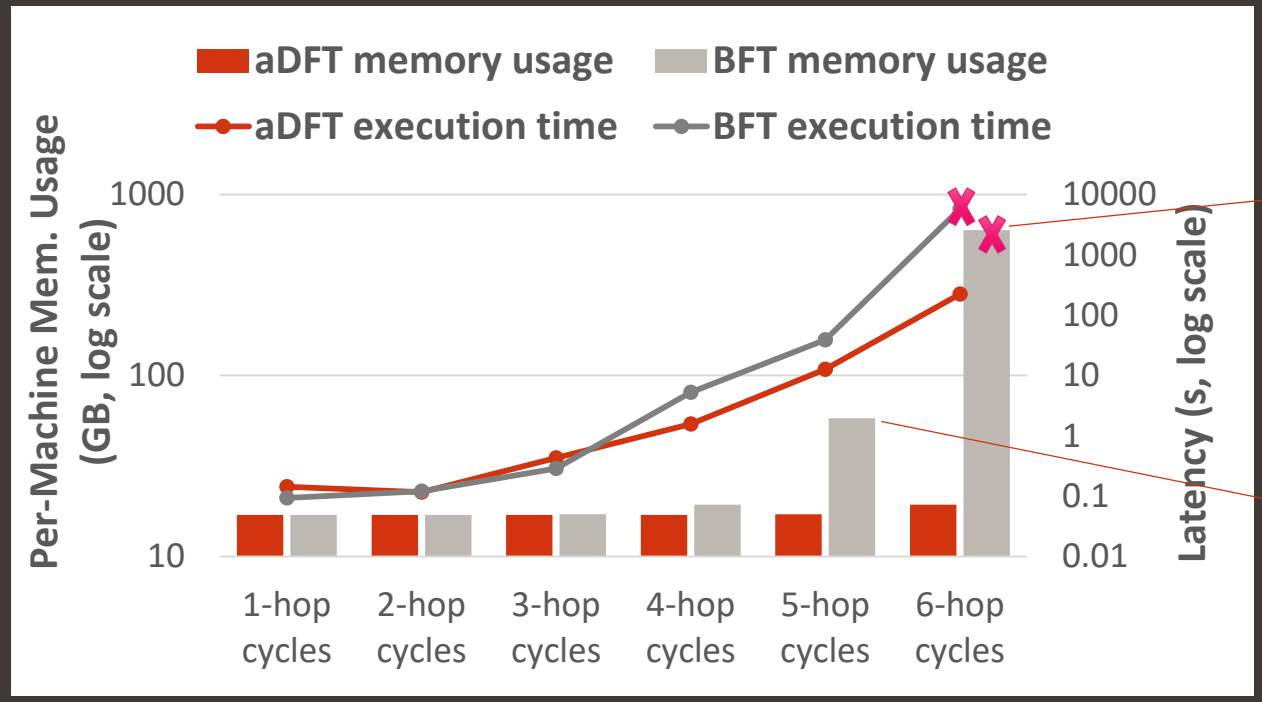
Query Scalability



LDBC 100 Social Graph (283M vertices, 1.78B edges) and Queries
PGX.D from 1 to 16 machines

Memory Consumption Control

Comparison against full BFS (bulk-synchronous)



8 * 768GB = 6TB is not enough

6 * 60 = 360GB

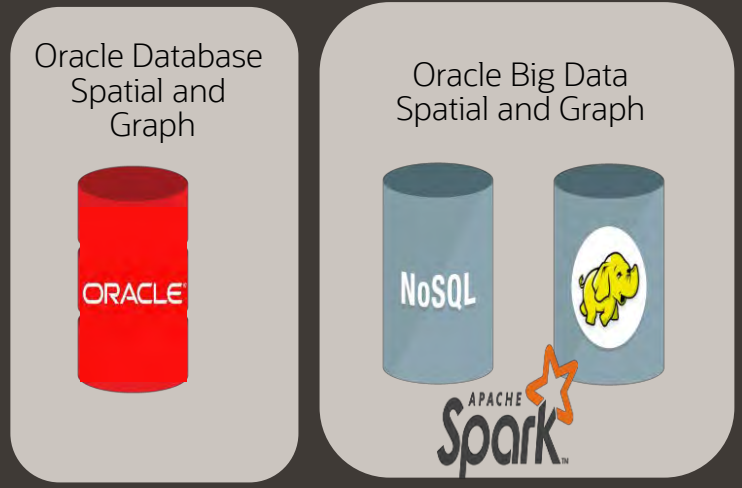
Matching cycles, Google web graph (875K vertices and 5.1M edges), 8 machines

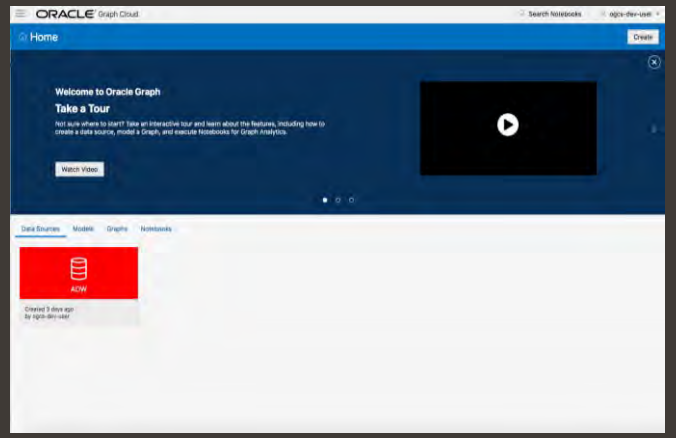
Graph Technology in Oracle

As individual products
- Oracle Database Spatial and Graph
- Oracle Big Data Spatial and Graph

As part of applications or solutions
- FCC Studio for FSGBU

As Cloud service (in preparation)
- Oracle Graph Cloud Service





Distributed In-memory Graph Query not part of product yet.
(Working towards it)

ORACLE

Graph at OOW and Code One 2019

View this list at bit.ly/SpatialGraphOOW19

Sessions

Date/Time	Title	Location
Tuesday, Sept. 17		
8:45 a.m. - 10:45 a.m.	**Using Graph Analytics for New Insights [TUT4328]**	Moscone South - Room 204
11:15 a.m. - 12:00 p.m.	**New Tools for the Fight Against Financial Crime [CON6222]**	Moscone West - Room 3004
12:30 p.m. - 1:15 p.m.	**Using Graph Analysis and Fraud Detection in the Fintech Industry [Paysafe customer session]**	Moscone South - Room 152C
12:30 p.m. - 1:15 p.m.	**Blazing-Fast Distributed Graph Query Processing: 100x as Fast as Spark [DEV3712]**	Moscone South - Room 307
3:15 p.m. - 4:00 p.m.	**Introducing Oracle Graph Cloud: Automating Graph Analysis [TRN4754]**	Moscone South - Room 159B
6:00 p.m – 6:45 p.m.	Towards Speech to Text for Arabic dialect on OCI [DEV 3862] (Lab's talk)	Moscone South - Room 313

Graph at OOW and Code One 2019

View this list at bit.ly/SpatialGraphOOW19

Sessions

Date/Time	Title	Location
Wednesday, Sept. 18		
9:00 a.m. - 9:45 a.m.	Achieve Automated Zero-Trust Security for SaaS Applications on Oracle Cloud Infrastructure [PRO5360] (brief mention)	Moscone South (Esplanade Ballroom)
10:00 a.m. – 10:45 a.m.	**Graph Databases and Analytics: How To Use Them [TRN4755]**	Moscone South - Room 152C
10:00 a.m. – 10:45 a.m.	**Setting Up Modern Anti-Money-Laundering Solutions to Service Wholesale [CON6223]**	Moscone West - Room 3004
11:15 a.m. - 12:00 p.m.	**Demystifying Graph Analytics for the Nonexpert [CON5503]**	Moscone South - Room 156B
1:30 p.m. - 2:15 p.m.	**Traversing and Querying Graphs with PGQL and Gremlin with Oracle Spatial and Graph [DEV4084]**	Moscone South - Room 202

Meet the Experts

At the Code One Groundbreakers Hub, Moscone South Level 1

Wednesday, Sept. 18		
1:30 pm - 2:20 pm	**Graph Database and Analysis**	Lounge C, Code One Groundbreakers Hub, Moscone South level 1
2:30 pm - 3:20 pm	**Graph Cloud Service: Automating Graph Analysis**	

Summary

- Graph Model and Graph Query
- Distributed graph query processing with Asynchronous traversal
  - Significant performance improvements over open-source solutions
  - With low memory consumption
- Technology from Oracle Labs
  - Working towards productization*

**Currently not included in shipping product - Safe Harbor terms apply*

Safe Harbor

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.