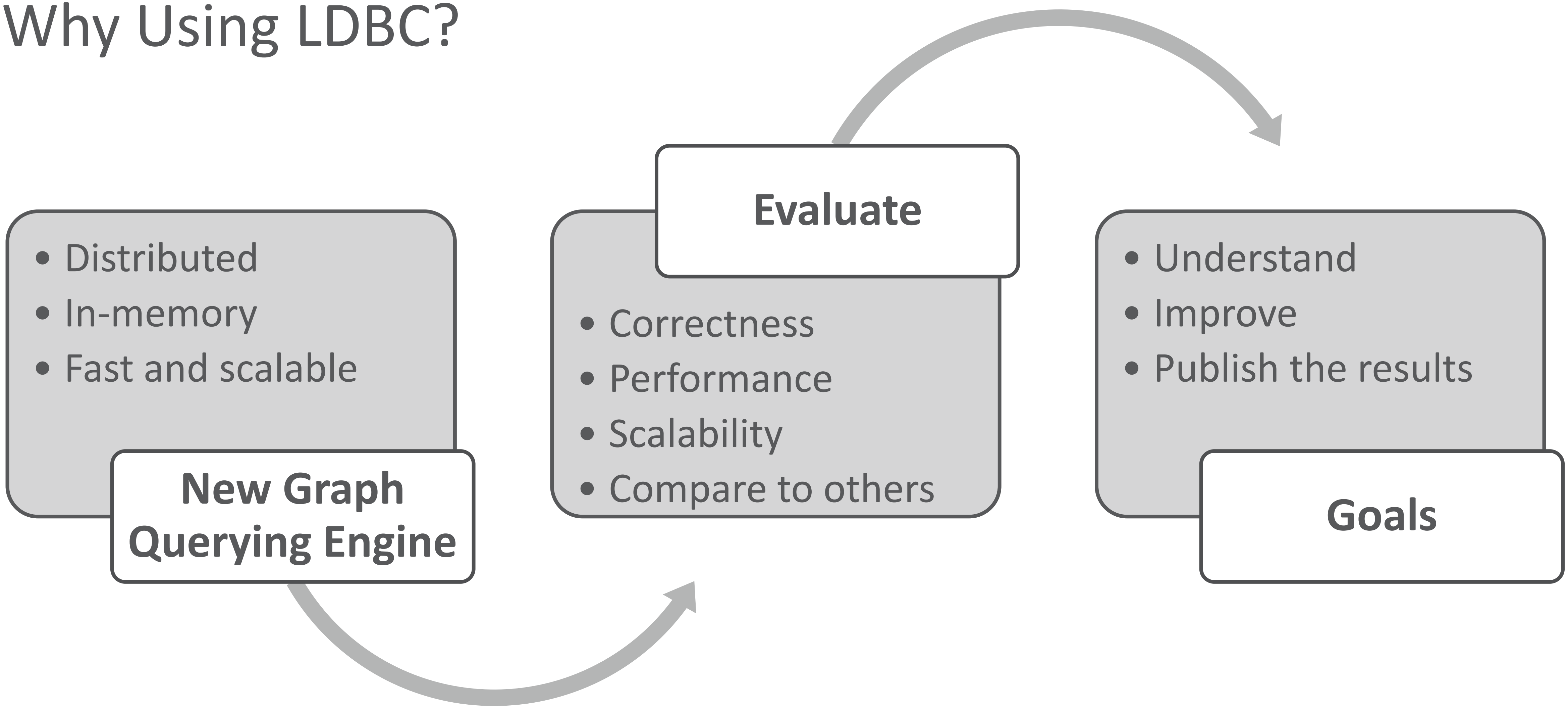
Evaluating a New Distributed Graph Query Engine with LDBC: Experiences and Limitations

Vasileios Trigonakis <vasileios.trigonakis@oracle.com>
Principal Researcher
Oracle Labs, PGX

Safe Harbor Statement

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Why Using LDBC?



- Distributed
- In-memory
- Fast and scalable

New Graph Querying Engine

Evaluate

- Correctness
- Performance
- Scalability
- Compare to others

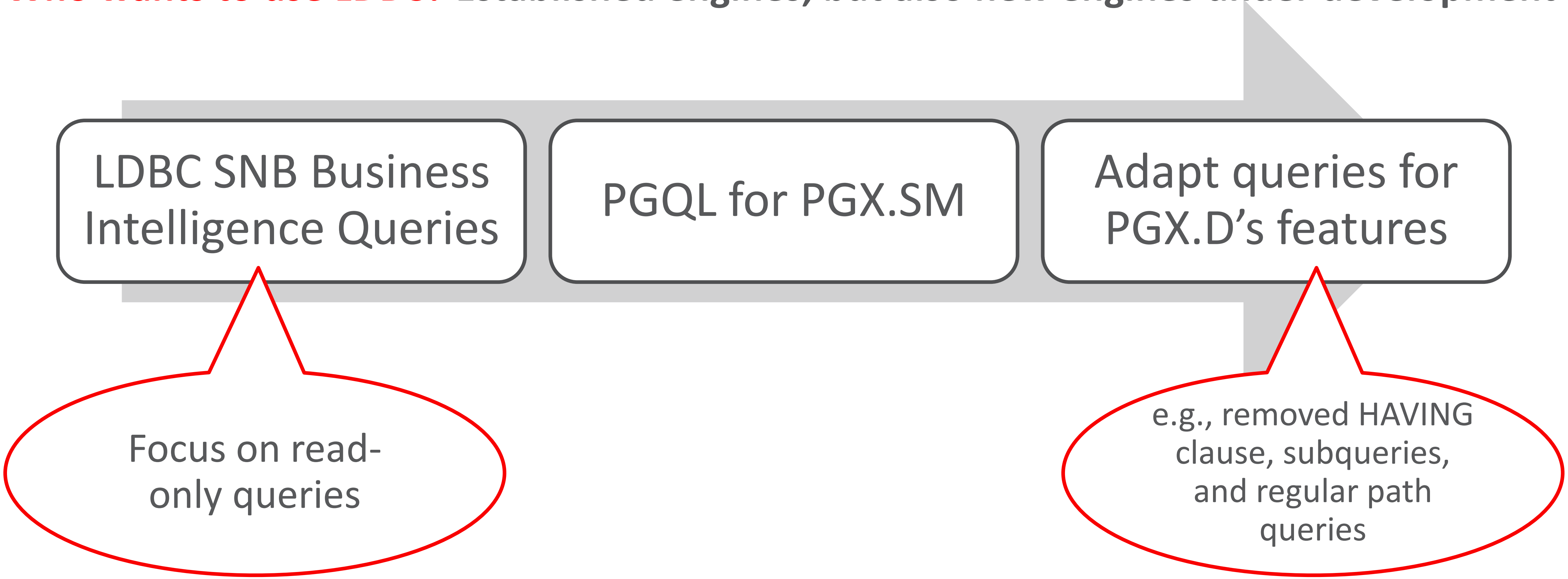
- Understand
- Improve
- Publish the results

Goals

Use LDBC because it is standardized and queries/graphs available for many engines

Using LDBC with PGX Distributed (PGX.D)

Who wants to use LDBC? Established engines, but also new engines under development



LDBC SNB Business Intelligence Queries

PGQL for PGX.SM

Adapt queries for PGX.D's features

Focus on read-only queries

e.g., removed HAVING clause, subqueries, and regular path queries

Other new-ish engines (e.g., Apache Spark GraphFrames) also need this last step

Outline – Experiences and Limitations

1. Query Complexity
2. Graph- vs. Relational-Friendly Queries
3. Query Size And Patterns
4. A Wishlist and Conclusions

Query Complexity

5 out of 25

Query #	Missing Feature
6	subquery
8	subquery + NOT EXISTS
11	subquery + NOT EXISTS
12	HAVING
14	regular path query (<-/:path*/)
15	HAVING + subquery
20	regular path query (<-/:path*/-)
21	subquery
22	subquery + EXISTS

- Started from 15 out 25
- Queries 2, 4, 17, 23, 24: Path queries with GROUP BY and ORDER BY
- For the rest: Removed missing features

Problems

1. Breaking the query semantics
2. Complexity
    1. Change PGQL, SQL, Cypher, and GraphFrames motifs)
    2. Confirm correctness, repeat

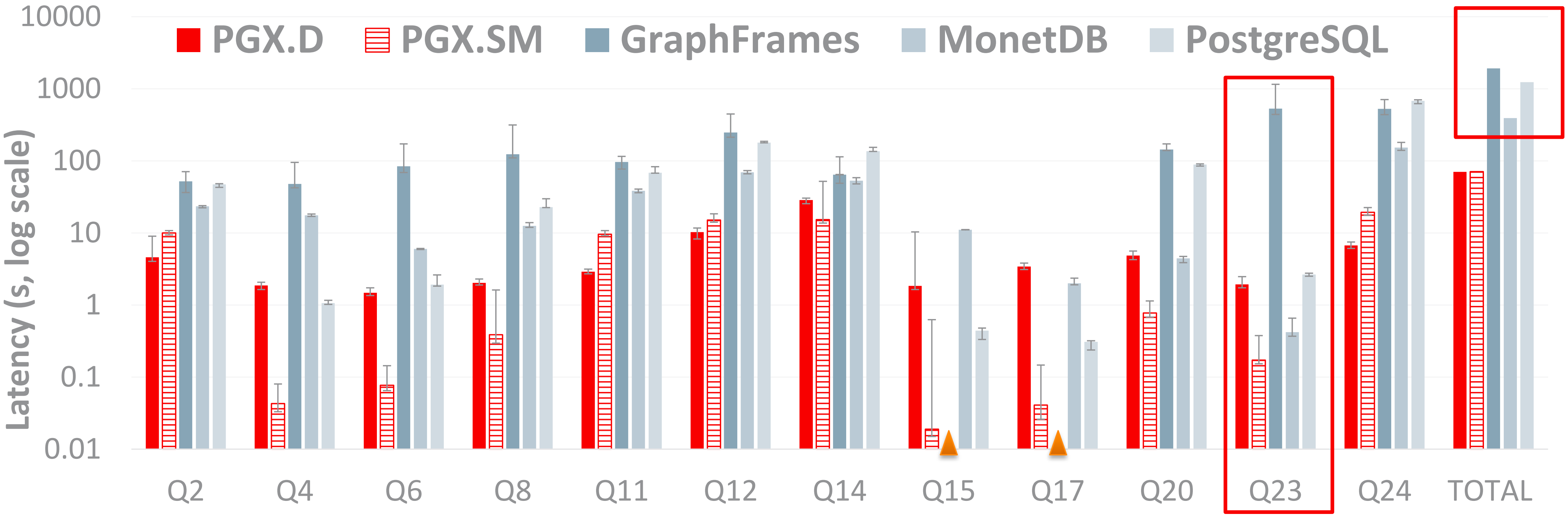
LDBC queries can be challenging to use for evaluating a new query engine

ORACLE

Graph- vs. Relational-Friendly Queries

▲ missing feature

- LDBC 100 SNB Graph (283M vertices, 1.78B edges)
- PGX.D and GraphFrames with 8 machines



The SQL engines do quite well. Similar results for other graph engines.

ORACLE

Q23 in PGQL and SQL

PGQL

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

SQL

```
SELECT COUNT(*) AS messageCount, ...
  FROM place pco, place pci, person p, message m, place dest
 WHERE pco.pl_placeid = pci.pl_containerplaceid
   AND pci.pl_placeid = p.p_placeid
   AND p.p_personid = m.m_creatorid
   AND m.m_locationid = dest.pl_placeid
   AND pco.pl_name = 'Egypt' AND NOT m.m_locationid = pco.pl_placeid
 GROUP BY m.m_creationdate, dest.pl_name
 ORDER BY messageCount DESC, dest.pl_name, m.m_creationdate
```

Very clean joins between rather small tables

Q23 Breakdown – LDBC 100 (283M vertices, 1.78B edges)

```
SELECT COUNT(msg) AS messageCount, …
MATCH (person:person)<-[:hasCreator]-(msg:post|comment)-[:isLocatedIn]->dst:country),
      (person)-[:isLocatedIn]->(city:city)-[:isPartOf]->(homeCountry:country)
WHERE homeCountry.name = 'Egypt' AND homeCountry <> dst
GROUP BY message.creationDate, destination.name
ORDER BY messageCount DESC, destination.name, message.creationDate
```

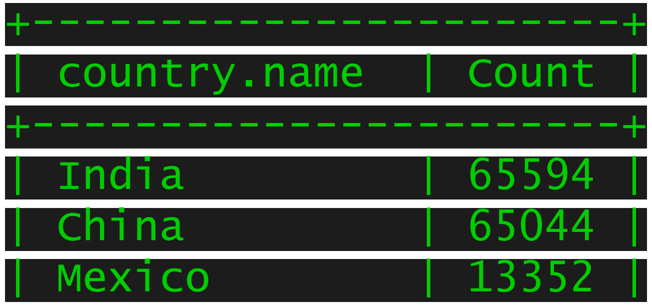
```
(person:person)<-[:hasCreator]-(msg:post|comment)-[:isLocatedIn]->dst:country),
(person)-[:isLocatedIn]->(city:city)-[:isPartOf]->(homeCountry:country)
```

Egypt 75224
All 10132079

```
(person)-[:isLocatedIn]->(city:city)-[:isPartOf]->(homeCountry:country)
```

Egypt 3351

```
SELECT country.name, COUNT(*) AS personCount
MATCH (:person)-[:isLocatedIn]->(:city)
        -[:isPartOf]->(country:country)
GROUP BY country
ORDER BY COUNT(*) DESC
```



```
+-----------------------+
| country.name  | Count |
+-----------------------+
| India         | 65594 |
| China         | 65044 |
| Mexico        | 13352 |
```

Long pattern, but with little data in most parts

Recurring Patterns

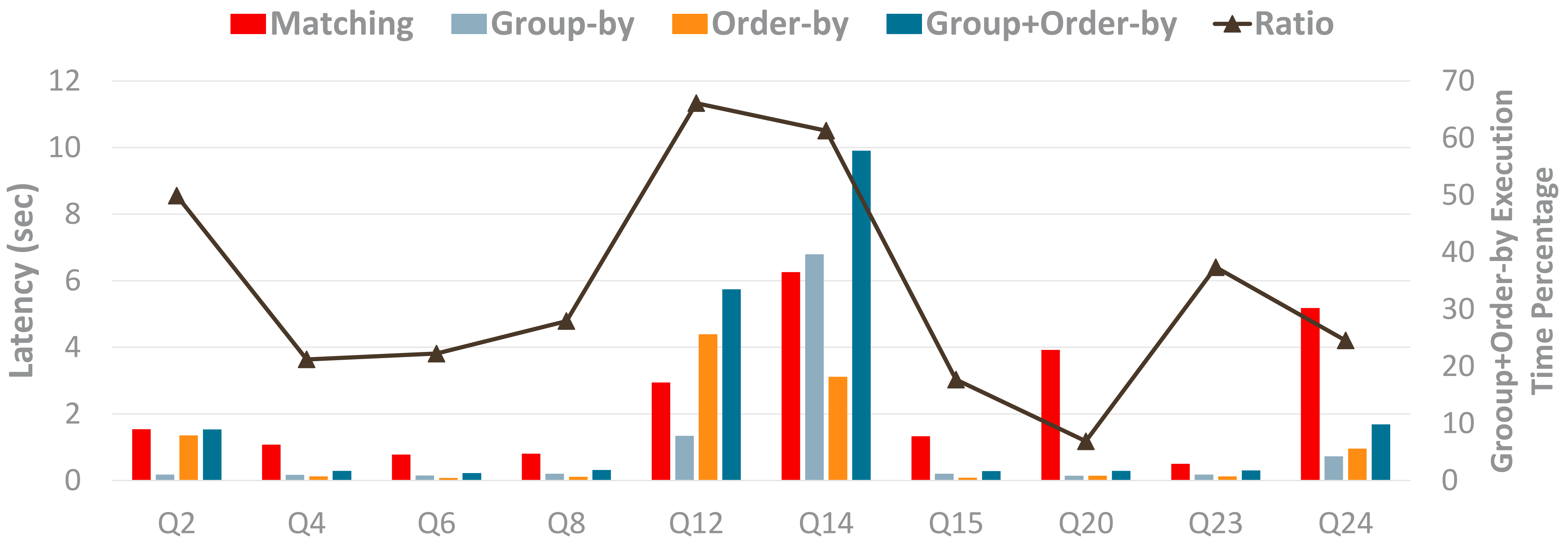
Query #	Pattern
2, 4, 11, 15, 17, 23, 24	(country:country) <-[:isPartOf]- (city:city) <-[:isLocatedIn]- (person:person) with country.name filter
4, 6, 20, 23	Tag or tagClass filter
All but query 17	GROUP BY
All but query 17	ORDER BY
All	Fully labeled accesses

What relational databases are built to do well

Not so many intermediate results (i.e., small-ish queries)

Pattern Matching Time Compared to Group By / Order By

- LDBC 100 (283M vertices, 1.78B edges)



Many queries are GROUP-BY and ORDER-BY heavy

A Possible Wish List (1/2)

(that would have made our lives easier while developing / evaluating PGX.D)

- A set with simple(r) pattern matching queries
  - No dependence on subqueries and regular path queries
- A set with realistic larger queries
  - Can be partially achieved by removing filters
  - Could e.g., analyze cycles in posts and comments
- Maybe less dependence on GROUP BY and ORDER BY

A Possible Wish List (2/2)

(that would have made our lives easier while developing / evaluating PGX.D)

- Queries that leverage the (homogenous) property graph model
  - E.g., paths / cycles:

```
SELECT labels(a), labels(b), labels(c), COUNT(*)
MATCH (a)->(b)->(c)->(a) GROUP BY a, b, c
```

  - Could combine with algorithms, e.g., pagerank values
- Look at the distributed graph direction (chokepoint)
  - E.g., how does graph partitioning affect different queries?

Conclusions

- Standardized graph benchmarks are a necessity
- LDBC SNB is a great effort towards this direction
  - but not easy for new engines as it requires complex query constructs

→From our recent experience, we see the need for:

- simpler,
- still meaningful,
- varying size queries

that can stress single machine and distributed graph engines

Thank You! Contact: vasileios.trigonakis@oracle.com