Traversing and Querying Graphs with Gremlin and PGQL on Oracle Spatial and Graph

Sungpack Hong
Research Director

Oracle Labs

Martin Sevenich
Principal Researcher

Oracle Labs

Oskar van Rest
Principal Member of Technical Staff
Oracle Spatial and Graph

Martijn Dwars
Senior Researcher

Oracle Labs

Safe Harbor

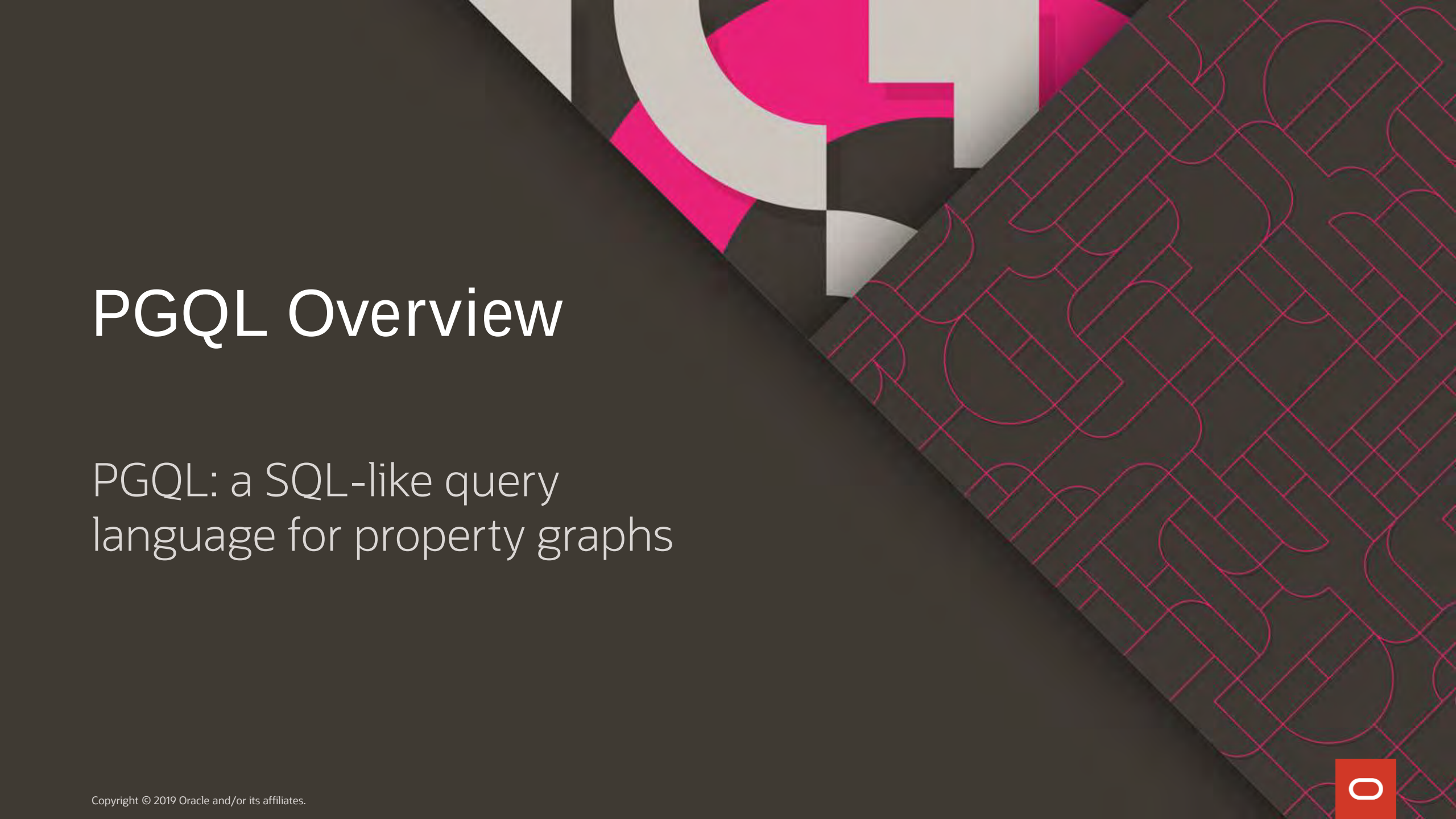
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, 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

- PGQL and Gremlin are two open source languages for querying and traversing property graphs. Although the two languages look very different on the surface, there are many parallels. This session introduces both languages, shows how common queries can be expressed in either language, and shows how to use both languages with Oracle Spatial and Graph. Finally, it shows how users can compile many of their existing Gremlin queries into PGQL queries.

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



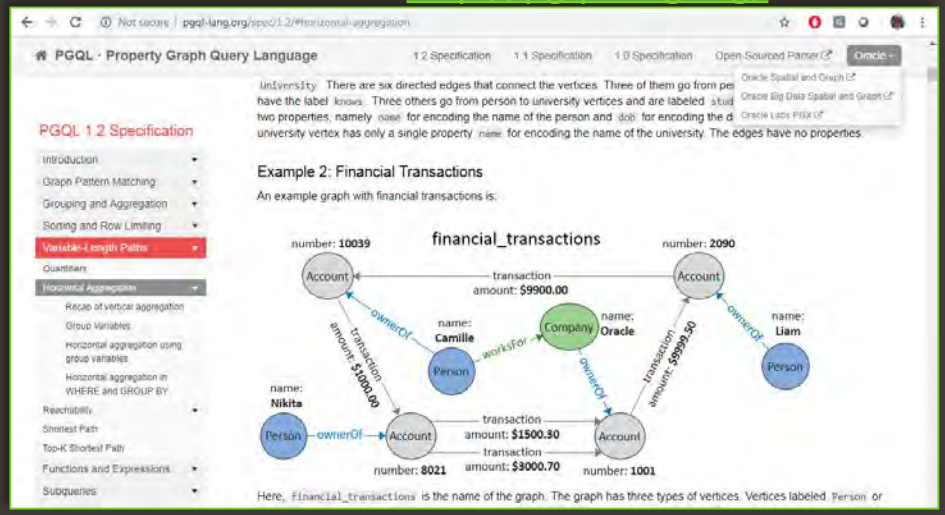
PGQL Overview

PGQL: a SQL-like query language for property graphs

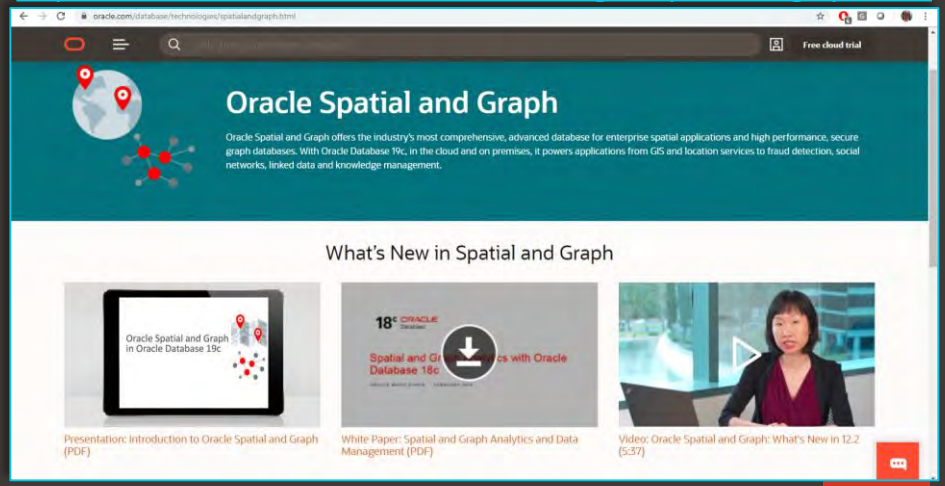
Property Graph Query Language (PGQL)

- PGQL is a "**graph pattern matching**" query language
  - Define a query as a high-level pattern and match it against graph data
- PGQL is **aligned to SQL** where possible
  - Provides familiarity for SQL users
- PGQL is part of **Oracle [Big Data] Spatial and Graph** products

PGQL home: http://pgql-lang.org/



Oracle Spatial and Graph home:
https://www.oracle.com/database/technologies/spatialandgraph.html

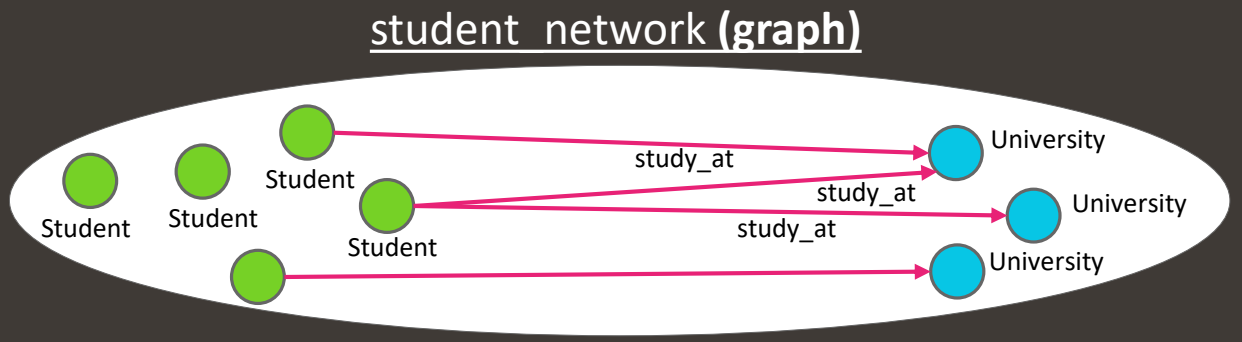



```
/* PGQL: find top-10 people with highest number of friends */
  SELECT n.firstName, COUNT(*) AS friendCount
    FROM socialNetwork
   MATCH (p:Person) -[:likes]-> (friend:Person)
GROUP BY p
ORDER BY friendCount DESC
   LIMIT 10
```

Property Graphs (1/2)

- A **property graph** consists of:
  - A set of **vertices**, each having:
    - A set of **properties** (key-value pairs)
    - A set of **labels** (character strings)
  - A set of **edges**, each having:
    - A **source vertex** and **destination vertex**
    - A set of **properties**
    - A set of **labels**
- Graphs are often aligned to the mental model that humans have of their data
  - Graphs make it **easy to analyze data**, even for non-programmers
  - Graphs can be **visualized naturally**, as graphs

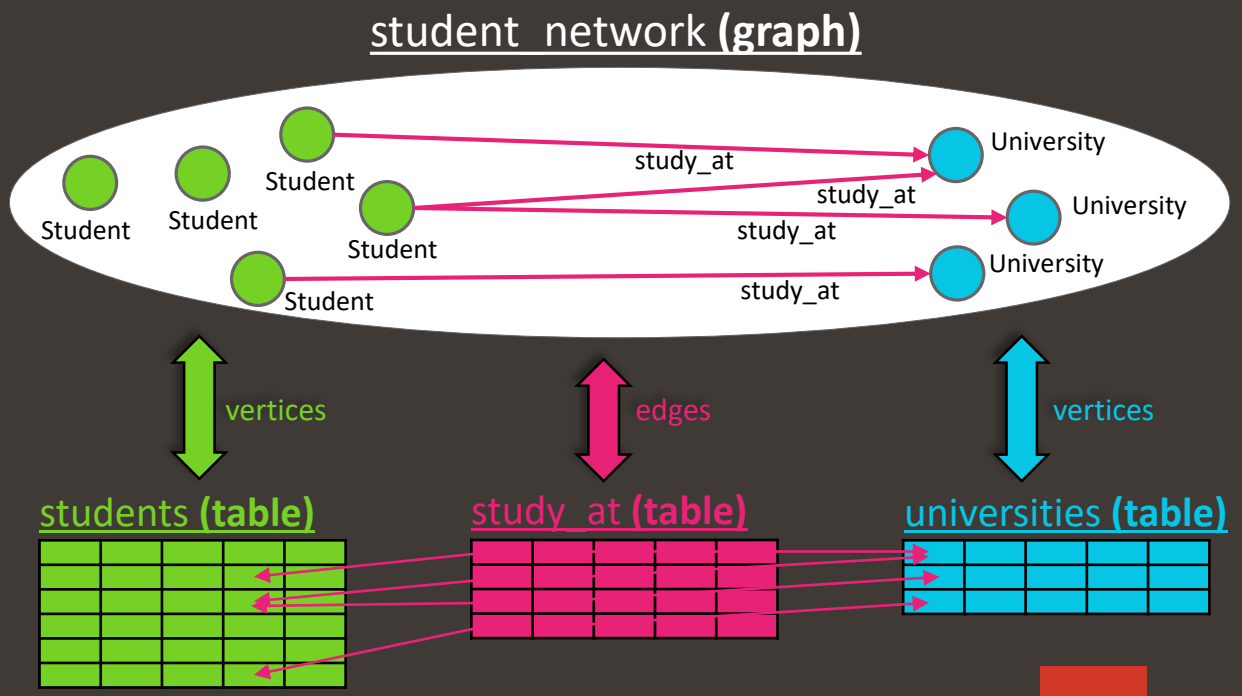
Example:
"student data" as **graph**



Property Graphs (2/2)

- Graphs **abstract** over how data is stored
  - Graph can be stored as **set of tables**, **XML** or **JSON** document, **set of triples**, **adjacency lists** (an efficient graph format), etc.
- Example graphs:
  - Bill of materials (product hierarchy)
  - Financial networks
  - Social networks
  - Physical networks
  - Knowledge graphs

Example:
"student data" as **graph**
vs. **set of tables**



Example: impact of network disruption

A power outage caused connectivity failures in an electric network (CLOSED). **Can devices still reach each other via alternative routes?**

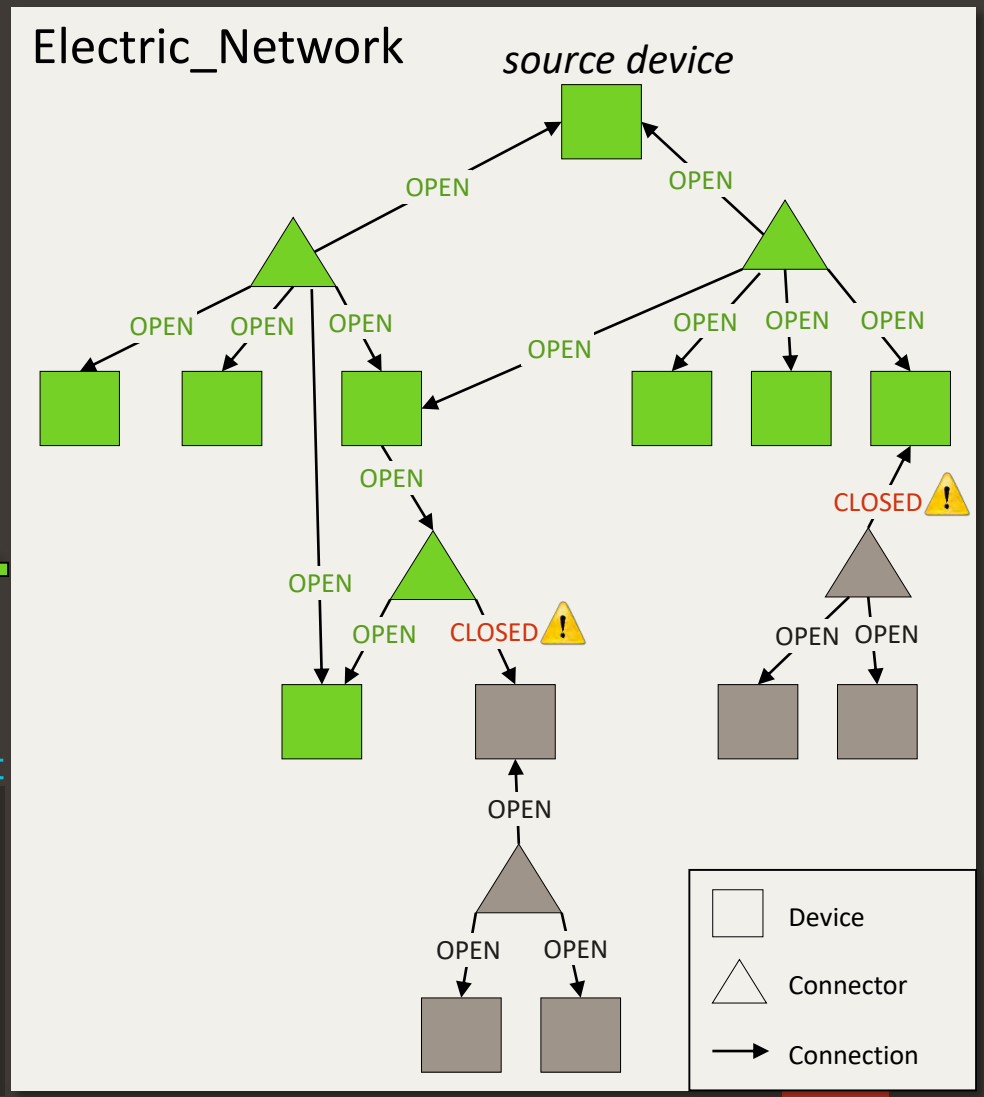
```
    PATH connects_to
         AS (from) <-[c1]- (connector) -[c2]-> (to)
         WHERE c1.status = 'OPEN'
           AND c2.status = 'OPEN'
  SELECT n.nickname, COUNT(m)
    FROM Electric_Network
   MATCH (n:Device) -/:connects_to*/-> (m:Device)
   WHERE java_regexp_like(n.nickname, 'Regulator')
     AND n <> m
GROUP BY n
ORDER BY COUNT(m) DESC, n.nickname
```

PGQL query

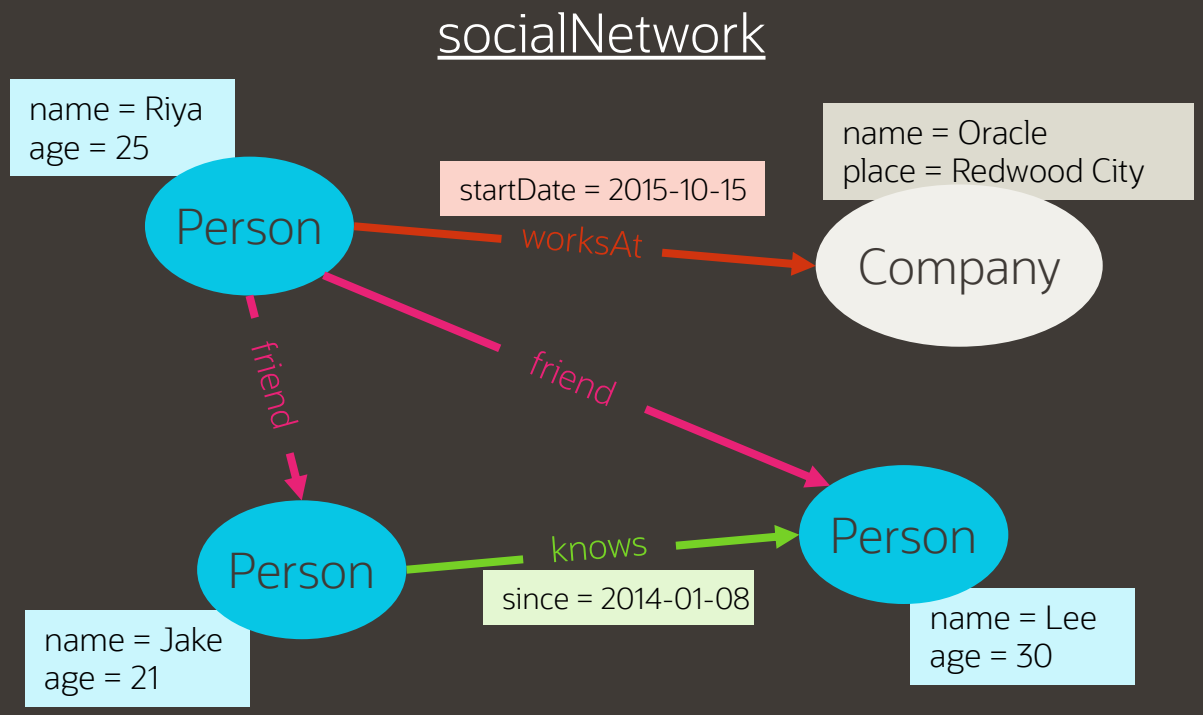
property graph

result set

n.nickname	COUNT(m)
Regulator, VREG2_A	1596
Regulator, VREG4_B	1537
Regulator, VREG4_C	1537
Regulator, HVMV_Sub_RegA	3
Regulator, HVMV_Sub_RegB	3



Graph pattern matching



PGQL query:
"find all people known by friends of Riya"

```
SELECT v1.name AS v1_name, v2.name AS v2_name,
     , v3.name AS v3_name, e.since AS knows_since
  FROM socialNetworkGraph
 MATCH (v1:Person) -[:friend]-> (v2:Person)
     , (v2) -[e:knows]-> (v3:Person)
 WHERE v1.name = 'Riya'
```

result set

v1_name	v2_name	v3_name	knows_since
Riya	Jake	Lee	2014-01-08

Graph queries vs. tabular queries

- **PGQL queries** are expressed very succinctly by means of **path patterns**
  - A path pattern consists of **vertex patterns** (e.g. `(n:Person)`) and **edge patterns** (e.g. `-[e:likes]->`)
- **Compared to joins in SQL**, path patterns increase the level of abstraction
  - Single edge pattern may correspond to multiple joins (between multiple pairs of tables)
  - Single edge pattern may correspond to recursive join (e.g. * for zero or more matches)
  - Etc.

```
WITH temp(device_id, device_name) AS (
  -- Anchor member:
  SELECT device_id, name
  FROM   Devices
  WHERE  name = 'Regulator, HVMV_Sub_RegB‘
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_Sub_RegB'
```

SQL query against a set of tables

equivalent

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

PGQL query against a property graph

Example **SQL** query vs. **PGQL** query

PGQL and SQL

PGQL uses syntax and semantics of **standard SQL where possible**:

- Same syntactic **query structure**:
  SELECT … FROM … WHERE … GROUP BY … HAVING … ORDER BY …
- Same **expressions and aggregations**:
  IS [NOT] NULL, CAST, CASE, IN, MIN/MAX/AVG/SUM, etc.
- Same (tabular) **subqueries**:
  Existential (EXISTS) subqueries, scalar subqueries, and more coming
- Same **data types** and expressions on those data types:
  DATE, TIME [WITH TIME ZONE], TIMESTAMP [WITH TIME ZONE], etc.
  EXTRACT(MONTH FROM my_date), etc.

Example: financial fraud detection

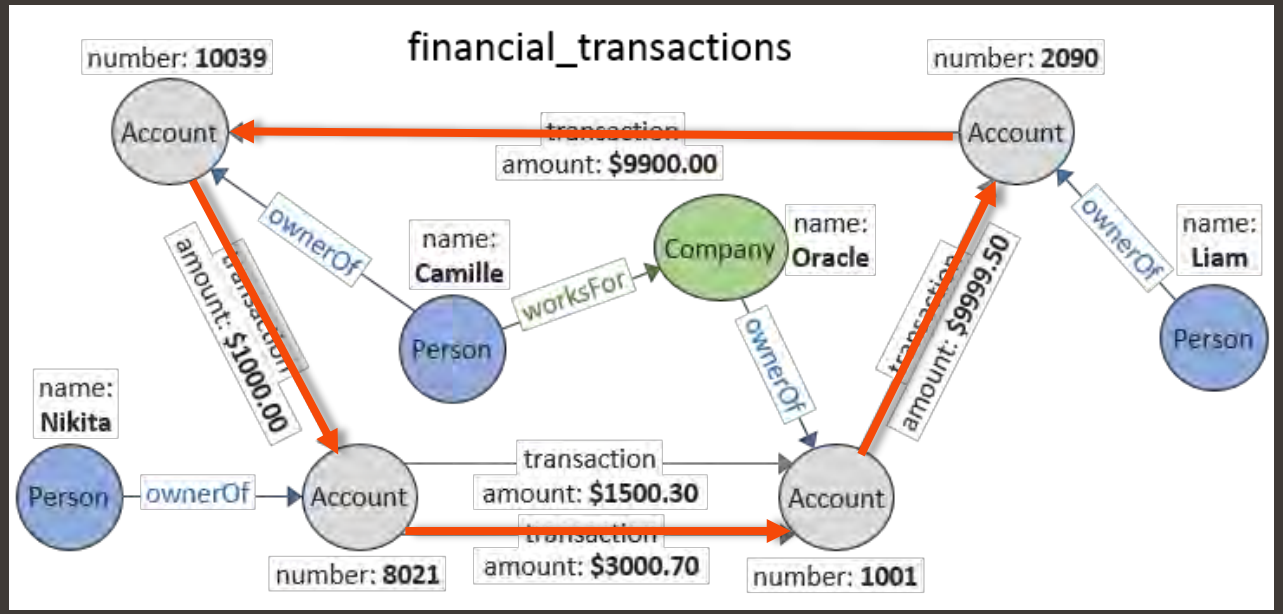
⚠ Find suspicious **circular payments** using a `SHORTEST` path query

Note: a payment is circular when it starts and ends in the same entity

PGQL query

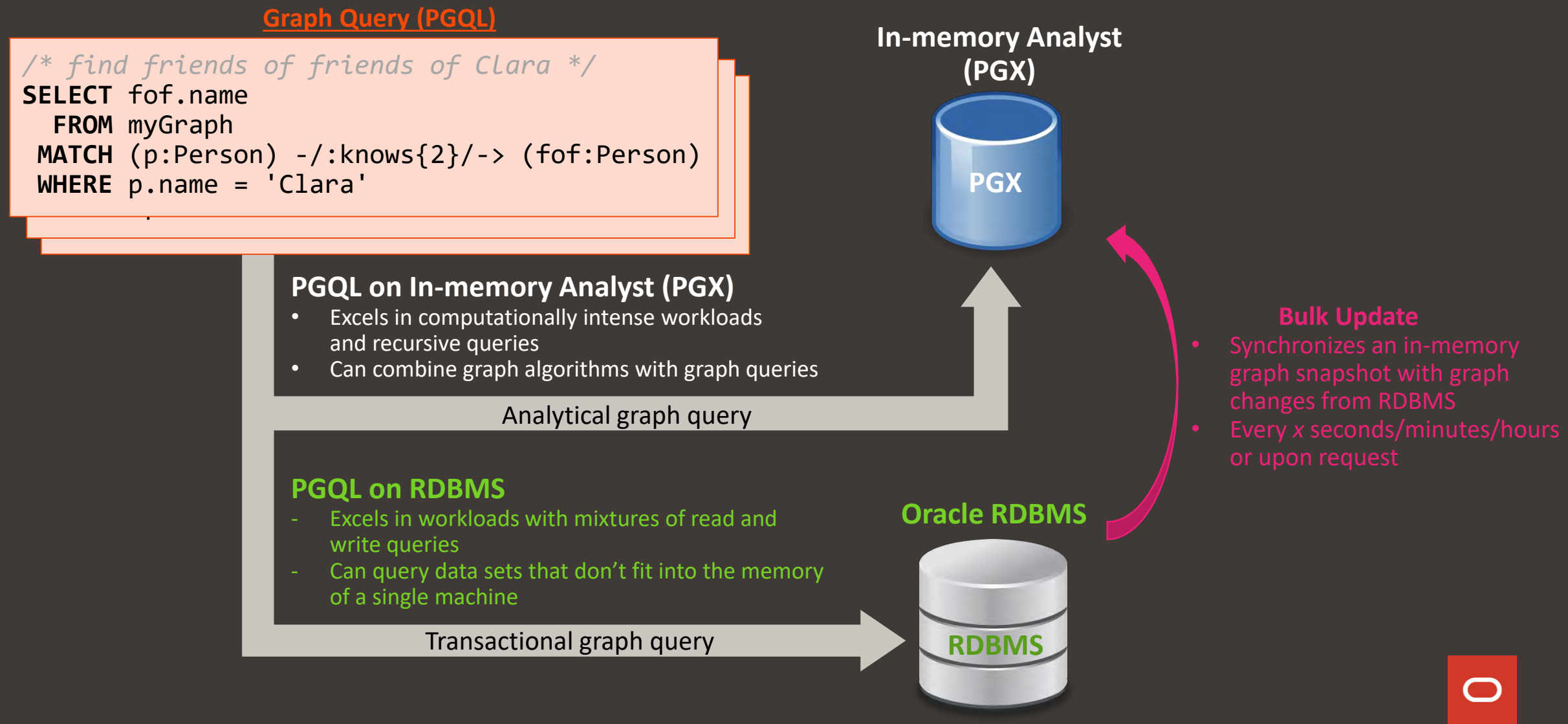
```
  SELECT COUNT(e) AS num_hops
       , SUM(e.amount) AS total_amount
       , ARRAY_AGG(e.amount) AS amounts_along_path
    FROM financial_transactions MATCH TOP 7 SHORTEST (
           (a:Account) -[e:transaction]->* (b:Account) )
   WHERE a.number = 10039 AND a = b
ORDER BY num_hops, total_amount
```

Property graph



num_hops	total_amount	amounts_along_path
0	<null>	<null>
4	22399.8	[1000.0, 1500.3, 9999.5, 9900.0]
4	23900.2	[1000.0, 3000.7, 9999.5, 9900.0] ⚠
8	44799.6	[1000.0, 1500.3, 9999.5, 9900.0, 1000.0, 1500.3, 9999.5, 9900.0]
8	46300.0	[1000.0, 1500.3, 9999.5, 9900.0, 1000.0, 3000.7, 9999.5, 9900.0]
8	46300.0	[1000.0, 3000.7, 9999.5, 9900.0, 1000.0, 1500.3, 9999.5, 9900.0]
8	47800.4	[1000.0, 3000.7, 9999.5, 9900.0, 1000.0, 3000.7, 9999.5, 9900.0]

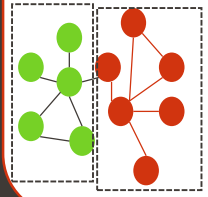
PGQL in Oracle Spatial and Graph (OSG)



Interaction with graph *algorithms*

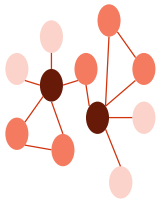
Oracle Spatial and Graph provides a rich set of **built-in graph algorithms**
(as part of the In-memory Analyst / PGX)

Detecting Components and Communities



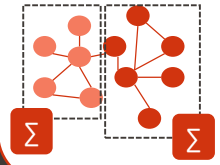
Tarjan's, Kosaraju's, Weakly Connected Components, Label Propagation (w/ variants), Soman and Narang's Sparcification

Ranking and Walking



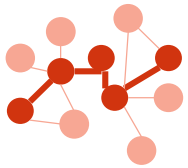
Pagerank, Personalized Pagerank, Betweenness Centrality (w/ variants), Closeness Centrality, Degree Centrality, Eigenvector Centrality, HITS, Random walking and sampling (w/ variants)

Evaluating Community Structures



Conductance,
Modularity
Clustering Coefficient
(Triangle Counting)
Adamic-Adar

Path-Finding



Hop-Distance (BFS)
Dijkstra's,
Bi-directional Dijkstra's
Bellman-Ford's

Link Prediction SALSA
(Twitter's Who-to-follow)

Other Classics Vertex Cover
Minimum Spanning-Tree
(Prim's)

- **PGQL graph queries** are often used in combination with **built-in graph algorithms**
  - Example:
    1. Run **Pagerank** algorithm ➔ each vertex gets a new "pagerank" property
    2. Run **PGQL** query to find the top-10 vertices with highest pageRank (`ORDER BY vertex.pagerank DESC`)

- *Upcoming: **PGX Algorithm** for defining your own graph algorithms*

Gremlin Overview

Gremlin: a graph-traversal language

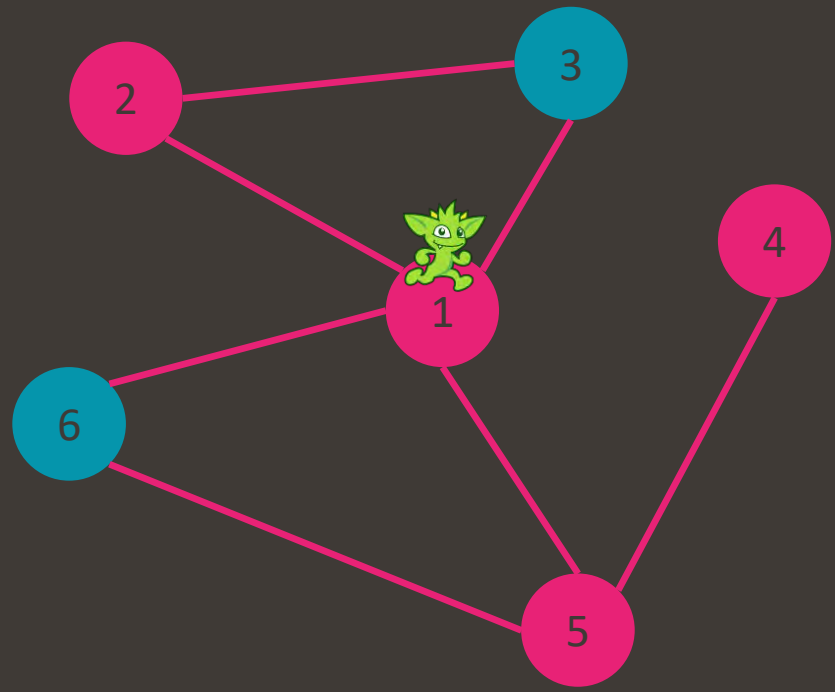
What is Gremlin?

- Gremlin is a functional data-flow language
- Gremlin is part of Apache TinkerPop
- Gremlin allows to express traversals on graphs
- Each traversal consists of a sequence of (nested) steps
  - A step performs an atomic operation on the data-stream

```
g.V(1)
  .out('red')
  .has('color', 'blue')
  .count()as
```

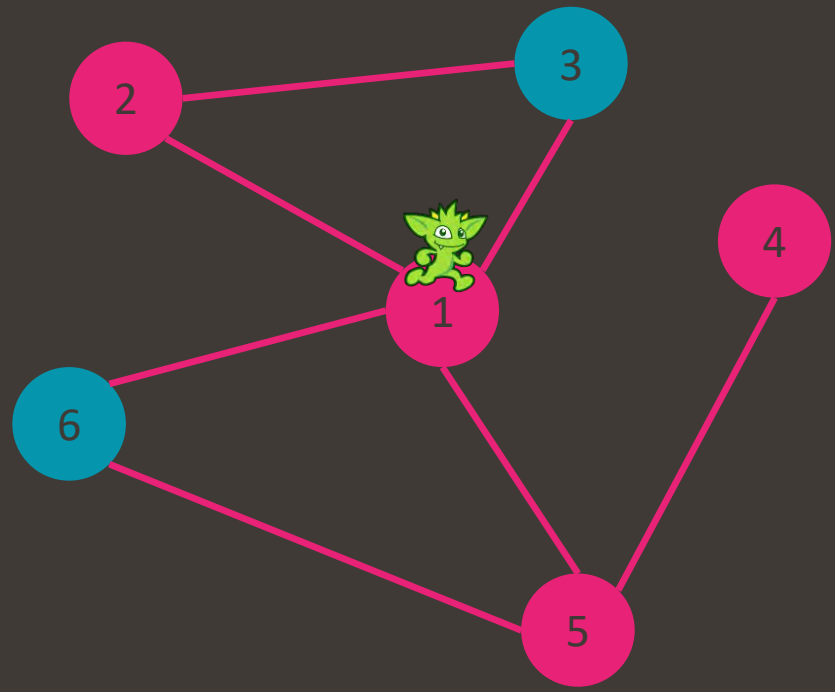
Gremlin Visualized

```
g.V(1)
 .out('red')
 .has('color', 'blue')
 .count()
```



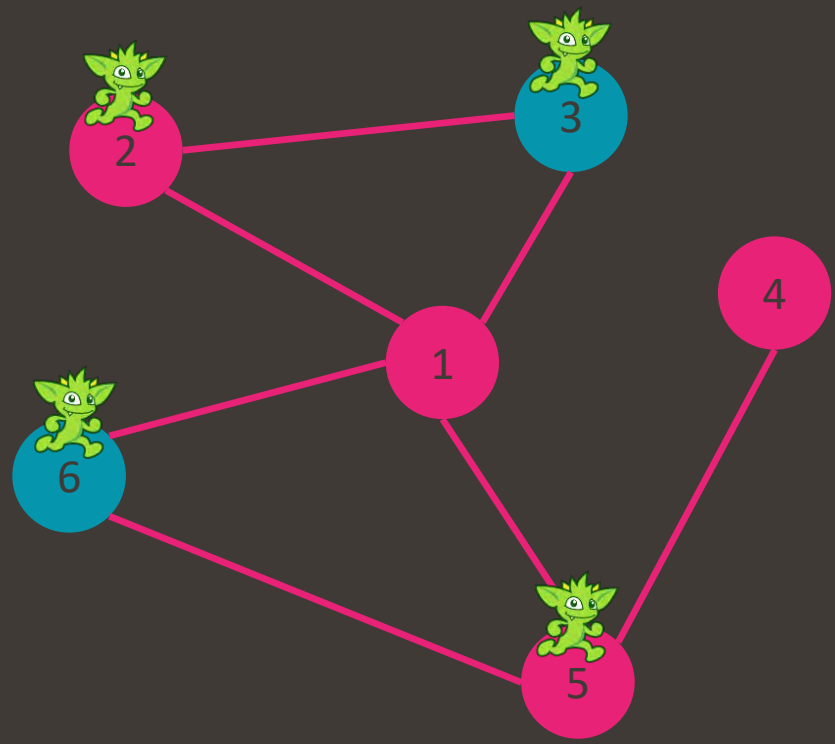
Gremlin Visualized

```
g.V(1)
 .out('red')
 .has('color', 'blue')
 .count()
```



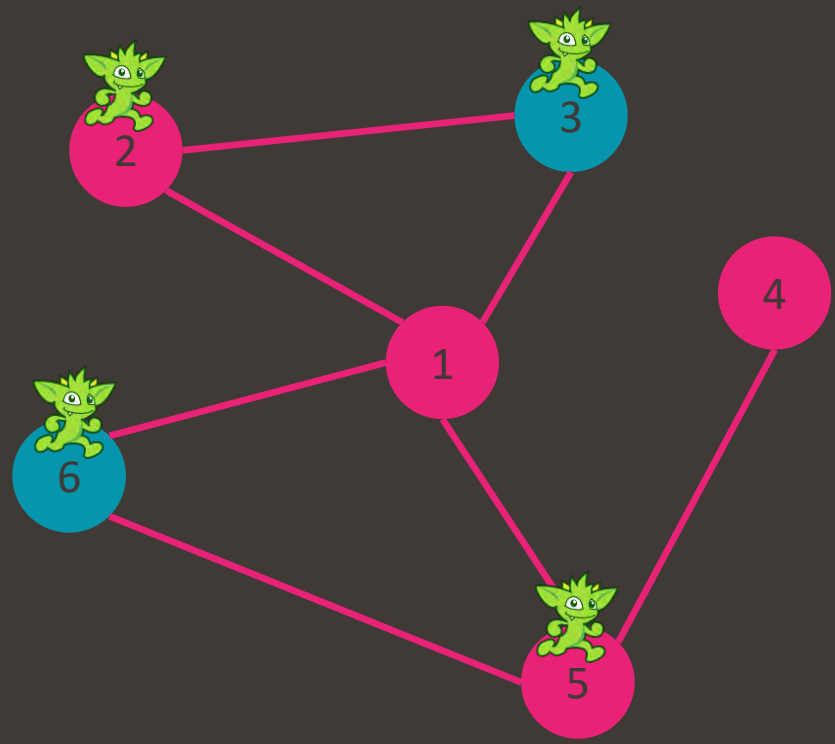
Gremlin Visualized

```
g.V(1)
 .out('red')
 .has('color', 'blue')
 .count()
```



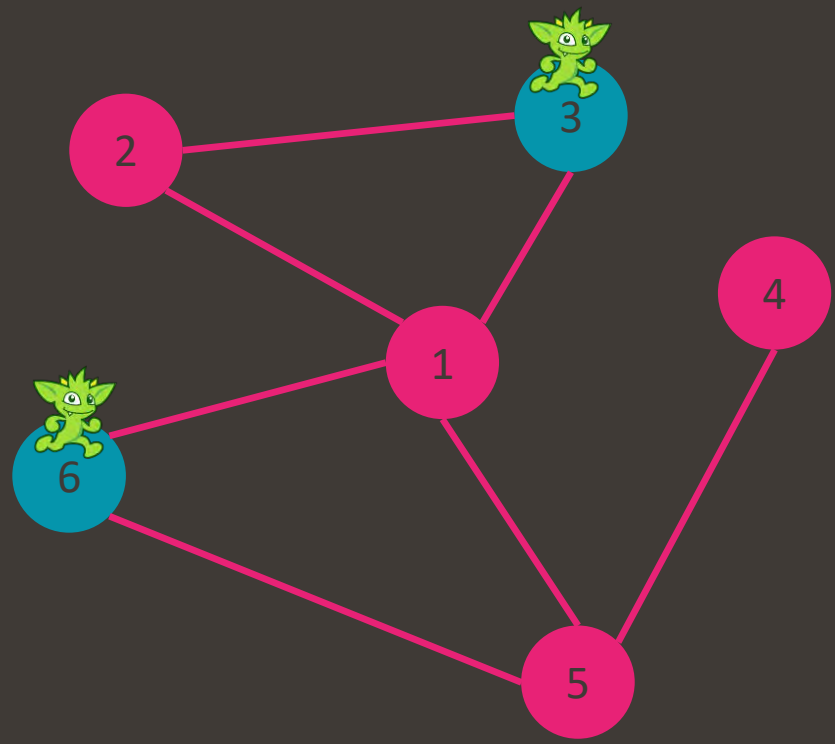
Gremlin Visualized

```
g.V(1)
 .out('red')
 .has('color', 'blue')
 .count()
```



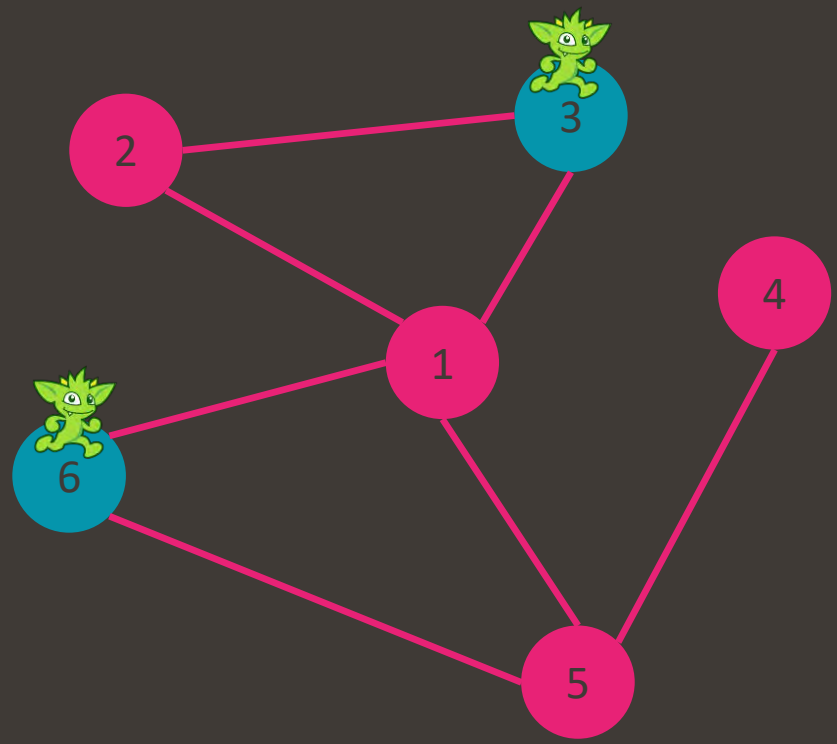
Gremlin Visualized

```
g.V(1)
 .out('red')
 .has('color', 'blue')
 .count()
```



Gremlin Visualized

```
g.V(1)
  .out('red')
  .has('color', 'blue')
  .count()
> 2
```



Gremlin Steps

Gremlin has a large variety of different steps

- Transform
  - Neighbors
  - Map to property value
- Filter
- Aggregations
- Side-effects
- Branch
  - If-then-else
  - Loops

PGQL vs. Gremlin

- PGQL is SQL-like
  - Familiar syntax & semantics
  - Declarative
- Query optimizer
  - PGQL can use its sophisticated query optimizer
  - No need to implement separate Gremlin optimizer / execution engine
- PGQL is easier to maintain

PGQL vs. Gremlin

Gremlin

```
g.V()
 .hasLabel('Customer').as('c')
 .values('name').as('name')
 .outE('Bought').as('b')
 .values('discount').as('disc')
 .select('b')
 .values('quantity').as('quant')
 .select('b')
 .outV()
 .has('Product', 'category', eq('toy')).as('p')
 .values('price').as('price')
 .select('price', 'disc', 'quant')
 .map{it.get().price*(1-it.get().disc/100.0)*it.get().quant}
   .as('sale_price')
 .select('name','sale_price')
 .group().by('name').by(sum())
```

users have to always keep the current context in mind

PGQL vs. Gremlin

PGQL

```
SELECT
  c.name, sum(p.price*(1-b.discount/100.0)*b.quantity)
MATCH
  (c:Customer) -[b:Bought]-> (p:Product)
WHERE
  p.category = 'toy'
GROUP BY
  c.name
```


Research & development

From Gremlin to PGQL

From Gremlin to PGQL

- Experimental feature developed by Oracle Labs
- Automatically convert a Gremlin traversal to a PGQL query
- Users can write Gremlin but use the PGQL execution engine

From Gremlin to PGQL

- Motivation:
  - No need to reinvent the wheel
    - Support Gremlin and PGQL with only the PGQL execution engine
    - Focus on improving one engine instead of two
    - Gremlin can benefit from existing PGQL query optimizer
  - Help users transition from Gremlin to PGQL
    - keep existing Gremlin traversals
    - Implement new queries in PGQL
    - Execute both on the same engine
    - Provide tool to auto-translate existing traversals if needed

Translating Gremlin to PGQL

Workflow

Gremlin Query

```
g.out()
 .distinct()
```


Translating Gremlin to PGQL

Workflow

Gremlin Query

```
g.out()
 .distinct()
```

Gremlin AST

```
[GraphStep(vertex,[]
),
VertexStep(OUT,verte
x),
DedupGlobalStep()]
```

Translating Gremlin to PGQL

Workflow

Gremlin Query

```
g.out()
 .distinct()
```

Gremlin AST

```
[GraphStep(vertex,[]
),
VertexStep(OUT,verte
x),
DedupGlobalStep()]
```

PGQL AST

```
Query(SelectClause(
  Distinct(),
rRef("v2")

attern(Vertices(["
```

Transformation rules

Translating Gremlin to PGQL

Workflow

Gremlin Query

```
g.out()
 .distinct()
```

Gremlin AST

```
[GraphStep(vertex,[]
),
VertexStep(OUT,verte
x),
DedupGlobalStep()]
```

PGQL AST

```
Query(SelectClause(
  Distinct(),
VarRef("v2")
 ),
GraphPattern(Vertices(["
v1", "v2"]),
[Edge("v1")])
)
```

PGQL Query

```
SELECT DISTINCT
  (v2)
MATCH
  (v1) -> (v2)
```

Transformation Rules

- Each Gremlin step has a transformation rule to PGQL

- The tool keeps track of the current context of the traversal
  - Each step might alter the context
  - Which alters the SELECT of the resulting query

Gremlin to PGQL: Transformations

Gremlin

```
g.V()
 .as("a")
  has("name", "gremlin")
  out("created")
  in("created")
  where(neq("a"))
  groupCount().by("title")
```

PGQL

```
SELECT
  a
MATCH
  (a)  [:created] > (b)
WHERE
  a.name = "gremlin"
GROUP


BY
  c.title
```

Gremlin to PGQL: Transformations

Gremlin

```
g.V()
 .as("a")
 .has("name", "gremlin")
  .out("created")
  .in("created")
  .where(neq("a"))
  .groupCount().by("title")
```

PGQL

```
SELECT
  a
MATCH
  (a)  [:created] > (b)
WHERE
  a.name = "gremlin"
GROUP


BY
   c.title
```

Gremlin to PGQL: Transformations

Gremlin

```
g.V()
  .as("a")
  .has("name", "gremlin")
  .out("created")
  .in("created")
  .where(neq("a"))
  .groupCount().by("title")
```

The query result changes with the context

PGQL

```
SELECT
  b
MATCH
  (a) -[:created]-> (b)
WHERE
  a.name = "gremlin"
GROUP


BY
  c.title
```

Gremlin to PGQL: Transformations

Gremlin

```
g.V()
 .as("a")
 .has("name", "gremlin")
 .out("created")
 .in("created")
  where(neq("a"))
  groupCount().by("title")
```

PGQL

```
SELECT
  c
MATCH
  (a) -[:created]-> (b),
  (b) <-[:created]- (c)
WHERE
  a.name = "gremlin"
GROUP

BY
  c.title
```

Gremlin to PGQL: Transformations

Gremlin

```
g.V()
 .as("a")
 .has("name", "gremlin")
 .out("created")
 .in("created")
 .where(neq("a"))
 .groupCount().by("title")
```

PGQL

```
SELECT
  c
MATCH
  (a) -[:created]-> (b),
  (b) <-[:created]- (c)
WHERE
  c != a
AND
  a.name = "gremlin"
GROUP BY
  c.title
```

Gremlin to PGQL: Transformations

Gremlin

```
g.V()
 .as("a")
 .has("name", "gremlin")
 .out("created")
 .in("created")
 .where(neq("a"))
 .groupCount().by("title")
```

PGQL

```
SELECT
  c.title, COUNT(*)
MATCH
  (a) -[:created]-> (b),
  (b) <-[:created]- (c)
WHERE
  c != a
AND
  a.name = "gremlin"
GROUP BY
  c.title
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell>
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell>
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell> t(g.V().out())
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell> t(g.V().out())
$12 ==> "SELECT v1 MATCH (v0) -> (v1)"

jshell>
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell> t(g.V().out())
$12 ==> "SELECT v1 MATCH (v0) -> (v1)"

jshell> (g.V()
   ...>   .hasLabel("Customer").as("c")
   ...>   .values("name").as("name")
   ...>   .select("c")
   ...>   .out("Bought").as("b")
   ...>   .values("discount").as("disc")
   ...>   .select("b")
   ...>   .in()
   ...>   .hasLabel("Product")
   ...> )
```

```
martin@martin-arch:gremlin2pgx(pgql)$ /usr/lib/jvm/java-11-openjdk/bin/java -cp $(cat gremlin2pgx.shell/build/classpath.txt) oracle.gm.lang.gremlin2pgx.Shell
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell> t(g.V().out())
$12 ==> "SELECT v1 MATCH (v0) -> (v1)"

jshell> (g.V()
   ...>   .hasLabel("Customer").as("c")
   ...>   .values("name").as("name")
   ...>   .select("c")
   ...>   .out("Bought").as("b")
   ...>   .values("discount").as("disc")
   ...>   .select("b")
   ...>   .in()
   ...>   .hasLabel("Product")
   ...> )
$13 ==> [ GraphStep(vertex, []), HasStep([~label.eq("Customer")]), AsStep(c), PropertiesStep([name], value), AsStep(name), SelectOneStep(last, c), VertexStep(OUT, [Bought], vertex), AsStep(b), PropertiesStep([discount], value), AsStep(disc), SelectOneStep(last, b), VertexStep(IN, vertex), HasStep([~label.eq("Product")]) ]

jshell>
```

```
|  Welcome to JShell -- Version 11.0.4
|  For an introduction type: /help intro

jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell> t(g.V().out())
$12 ==> "SELECT v1 MATCH (v0) -> (v1)"

jshell> (g.V()
   ...>   .hasLabel("Customer").as("c")
   ...>   .values("name").as("name")
   ...>   .select("c")
   ...>   .out("Bought").as("b")
   ...>   .values("discount").as("disc")
   ...>   .select("b")
   ...>   .in()
   ...>   .hasLabel("Product")
   ...> )
$13 ==> [ GraphStep(vertex, []), HasStep([~label.eq("Customer")]), AsStep(c), PropertiesStep([name], value), AsStep
(name), SelectOneStep(last, c), VertexStep(OUT, [Bought], vertex), AsStep(b), PropertiesStep([discount], value), As
Step(disc), SelectOneStep(last, b), VertexStep(IN, vertex), HasStep([~label.eq("Product")]) ]

jshell> t(g.V()
   ...>    .hasLabel("Customer").as("c")
   ...>    .values("name").as("name")
   ...>    .select("c")
   ...>    .out("Bought").as("b")
   ...>    .values("discount").as("disc")
   ...>    .select("b")
   ...>    .in()
   ...>    .hasLabel("Product")
   ...> )
```

```
jshell> g.V().out()
$11 ==> [ GraphStep(vertex, []), VertexStep(OUT, vertex) ]

jshell> t(g.V().out())
$12 ==> "SELECT v1 MATCH (v0) -> (v1)"

jshell> (g.V()
   ...>   .hasLabel("Customer").as("c")
   ...>   .values("name").as("name")
   ...>   .select("c")
   ...>   .out("Bought").as("b")
   ...>   .values("discount").as("disc")
   ...>   .select("b")
   ...>   .in()
   ...>   .hasLabel("Product")
   ...> )
$13 ==> [ GraphStep(vertex, []), HasStep([~label.eq("Customer")]), AsStep(c), PropertiesStep([name], value), AsStep
(name), SelectOneStep(last, c), VertexStep(OUT, [Bought], vertex), AsStep(b), PropertiesStep([discount], value), As
Step(disc), SelectOneStep(last, b), VertexStep(IN, vertex), HasStep([~label.eq("Product")]) ]

jshell> t(g.V()
   ...>    .hasLabel("Customer").as("c")
   ...>    .values("name").as("name")
   ...>    .select("c")
   ...>    .out("Bought").as("b")
   ...>    .values("discount").as("disc")
   ...>    .select("b")
   ...>    .in()
   ...>    .hasLabel("Product")
   ...> )
$14 ==> "SELECT v2 MATCH (v0:Customer) -[:Bought]-> (v1), (v1) <- (v2:Product)"

jshell>
```

Supported Gremlin Features

map

- Constant
- Count
- Id
- Label
- Match
- Math
- Max
- Mean
- Min
- Order
- Project
- Select
- Sum

flatMap

- Vertex

filter

- And
- Dedup
- Has
- Is
- Limit
- Not
- Or
- Range
- Skip
- Where

sideEffect

- Group
- GroupCount
- …

Unsupported Gremlin Features

- Multiple input graphs
  - Rarely seen in real-life queries
- Graph mutation (insert / delete)
  - Will be supported in future PGQL version
- Control flow / algorithmic computation
  - Oracle's Spatial & Graph offers the PGX Algorithm API
    - Intuitive Java API for implementing graph algorithms
    - Offer specialized features for each use-case
      - PGQL for queries
      - PGX Algorithm API for algorithms

Summary

- PGQL is Oracle's property graph query language
  - Powerful pattern matching query language
  - Has integration with graph algorithms
  - Part of Oracle [Big Data] Spatial and Graph
- Gremlin is a popular open source graph traversal language
- Research area: We can auto-transform Gremlin to PGQL*
  - Easier transition for users
  - Utilize benefits of PGQL

*Currently not included in shipping product – Oracle Safe Harbor terms apply

Would you like to know more?

Graph at OOW and Code One 2019

Sessions

Wednesday, Sept. 18		
10:00 am - 10:45 am	**Graph Databases and Analytics: How To Use Them [TRN4755]**	Moscone South - Room 152C
10:00 am - 10:45 am	**Setting Up Modern Anti-Money-Laundering Solutions to Service Wholesale [CON6223]**	Moscone West - Room 3004
11:15 am - 12:00 pm	**Demystifying Graph Analytics for the Nonexpert [CON5503]**	Moscone South - Room 156B
1:30 pm - 2:15 pm	**Traversing and Querying Graphs with PGQL and Gremlin with Oracle Spatial and Graph [DEV4084]**	Moscone South - Room 202

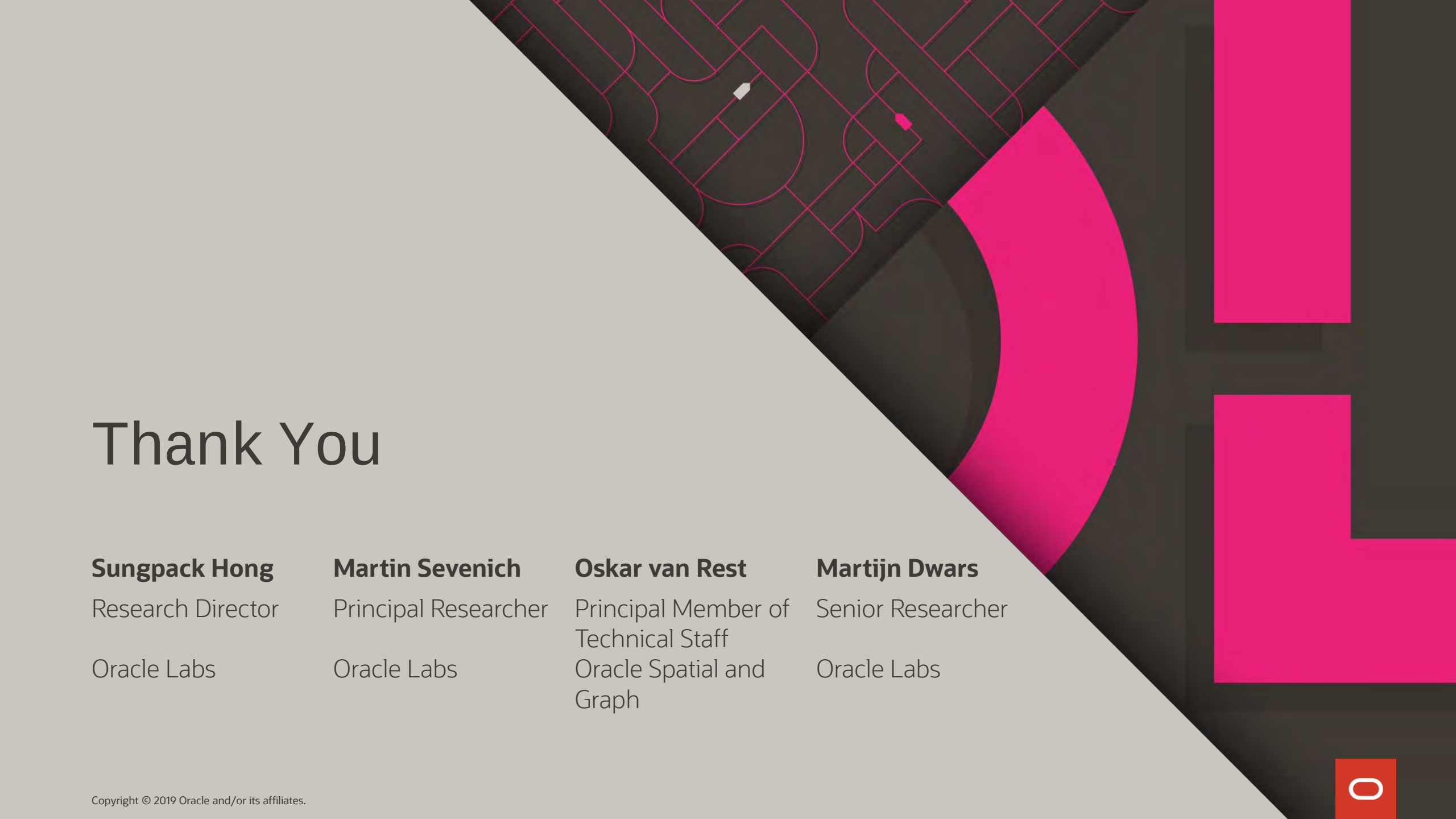
Meet the Experts At the Code One Groundbreakers Hub, Moscone South Level 1

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**	

Graph at OOW and Code One 2019

Date/Time	Title	Location
Monday 10:00 am – 4:00 pm Tuesday 10:30 am – 5:30 pm Wednesday 10:00 am– 4:30 pm	Spatial and Graph: Database, Analytics and Cloud	Moscone South Exhibit Hall (‘The Exchange’) Oracle Demogrounds > Data Management area > Kiosk # ODB-017

Questions?



Thank You

Sungpack Hong
Research Director

Oracle Labs

Martin Sevenich
Principal Researcher

Oracle Labs

Oskar van Rest
Principal Member of Technical Staff
Oracle Spatial and Graph

Martijn Dwars
Senior Researcher

Oracle Labs

Session Survey

Help us make the content even better. Please complete the session survey in the Mobile App.

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.

Break New Ground

San Francisco
September 16–19, 2019