# Performing OLAP over Graph Data: Query Language, Implementation, and a Case Study

Leticia Gómez, Alejandro Vaisman
Instituto Tecnológico de Buenos Aires, Argentina

Bart Kuijpers
Hasselt University and Transnational University of Limburg, Belgium

# Outline

- Introduction and motivation
- Data model
- OLAP over graph data
- Use case and implementation
- Conclusion

# Outline

- Introduction and motivation
- Data model
- OLAP over graph data
- Use case and implementation
- Conclusion

# Motivation

- Graphs DBs extensively used nowadays
  - Model different kinds of networks (social, sensor, etc.)
- Two models: Property graphs and RDF graphs
  - Can be reconciled (Hartig, 2014)
- Typical tool: Neo4J / Cypher
  - Property graph model
  - Widely used, medium-sized graphs
- For large graphs: graph processing frameworks, based on vertex-centric computation for parallel processing

# Motivation

- We want to use graphs for classic analysis and also for OLAP-like analysis (e.g, graph aggregation), and/or a combination of them
- The main question we try to answer here is

  What does OLAP on graph data mean?
- Existing proposals (IOO) do not answer this question
  - Address particular cases, typically binary relationships
  - Graph OLAP (Chen et al., Know. Inf. Sys, 2009), Graph Cube (Zhao et al., SIGMOD 2011), Pagrol (Wang et al., ICDE 2014)
  - Do not characterize OLAP on graphs
  - Define operators, do not address complex queries

# Motivation

– We need to understand the problem

– Actually, do we need OLAP on graphs?

- Find interesting use cases

– Can it be solved?

- A comprehensive data model
- Query language: what operations do we need?
- Simulate the typical OLAP operations on graphs
- Combine the powers of graphs and the cube metaphor

–Efficiency

- Can OLAP queries be executed in reasonable time?
- What do we need?
- Can this be compared against, e.g., ROLAP?

# Motivation

–Our contribution

- A formal data model based on multi-hypergraphs
- Formal definition of typical OLAP operators on graphs
- Uses cases aimed at showing where OLAP on graphs can be helpful
- Proof-of-concept implementation over Neo4J showing the plausibility of the approach
  - Not the best option for efficiency
  - Works with small/medium sized graphs
  - Easy to implement

# Outline

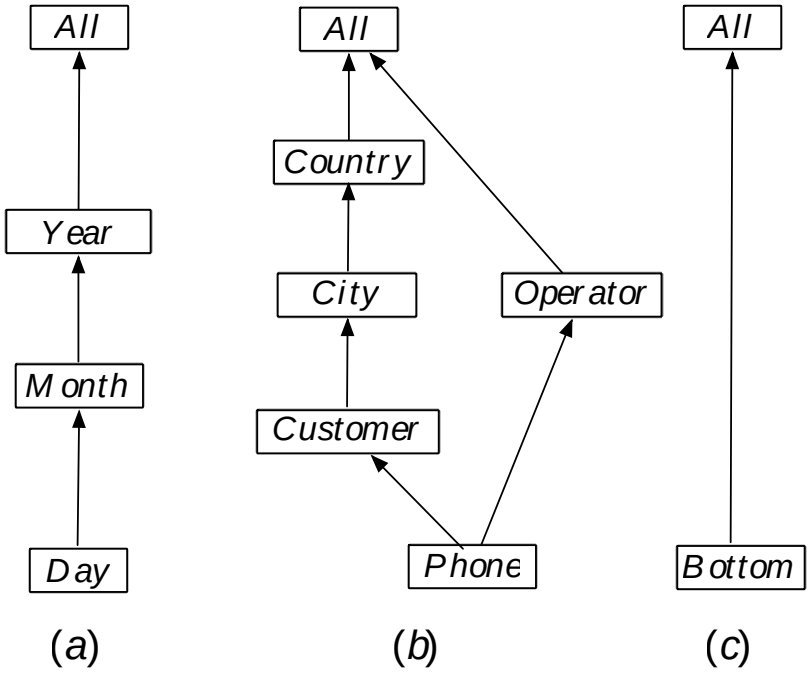
- Introduction and motivation
- Data model
- OLAP over graph data
- Use case and implementation
- Conclusion

# Data model

- Composed by
  - Background dimensions (as in typical OLAP)
    - Dimension schemas
    - Dimension instances
  - Base Graph
  - Graphoids
    - Aggregation of the base graph at different granularities
  - Base graph and graphoids are multi-hypergraphs
- Will use an example about group phone calls (not in the paper)

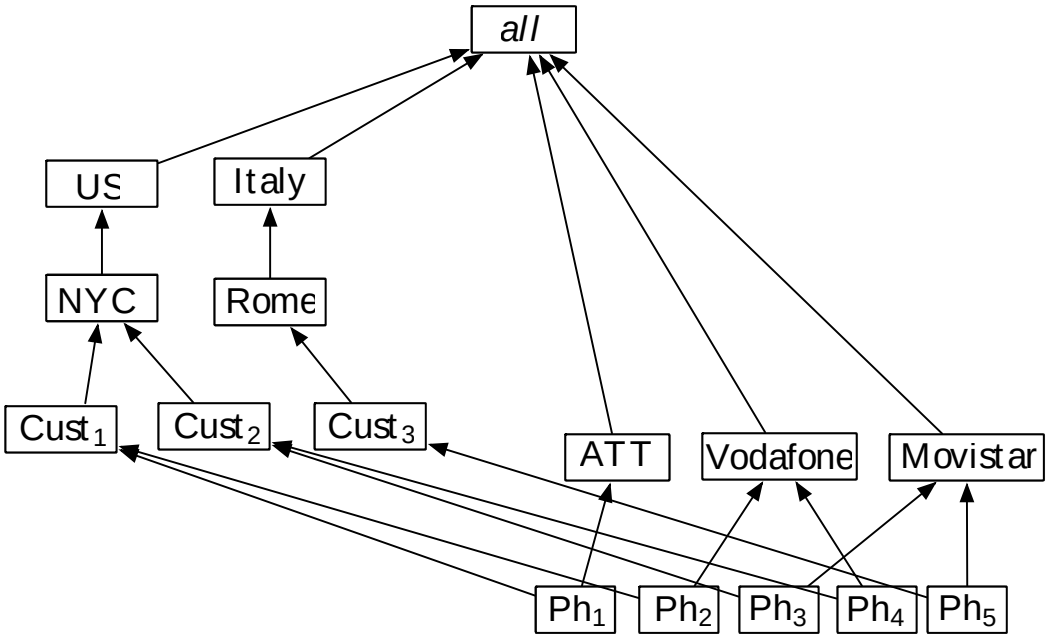
# Data model

- **Background Dimension Schemas**



# Data model

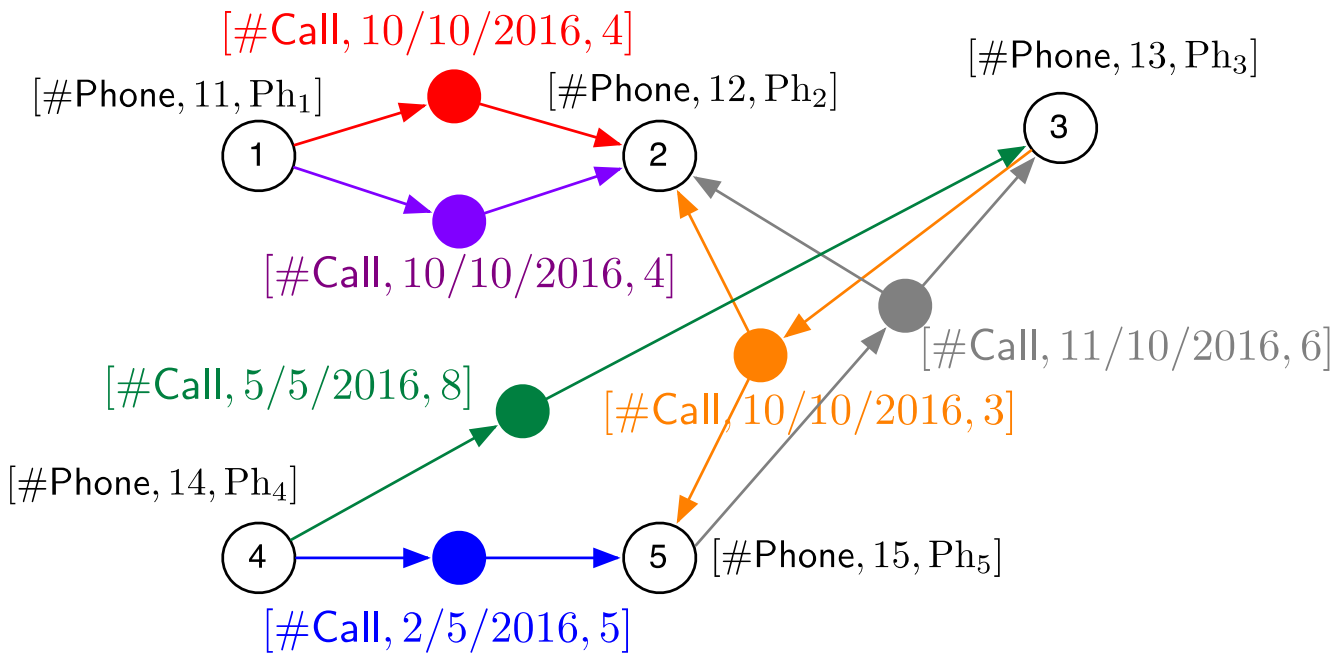
- **Dimension Phone: Instance**



# Data model

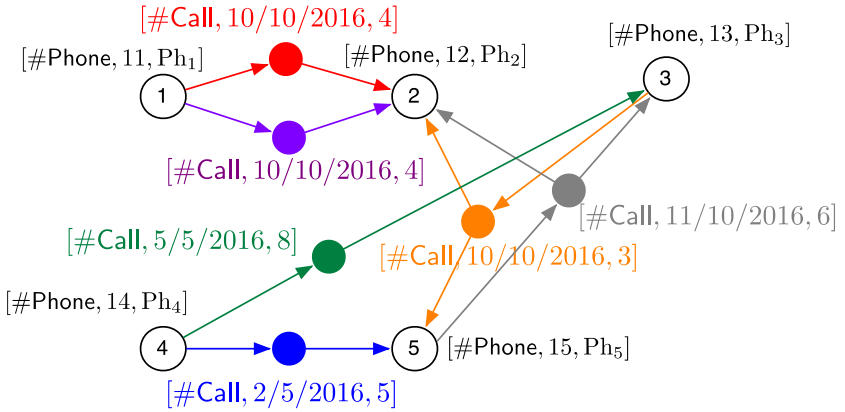
- Facts represented as multi-hypergraphs
- Example: Phone calls (point-to point, group calls)

  Base Graph: (Phone.Phone, Time.Day)-Graphoid



# Data model

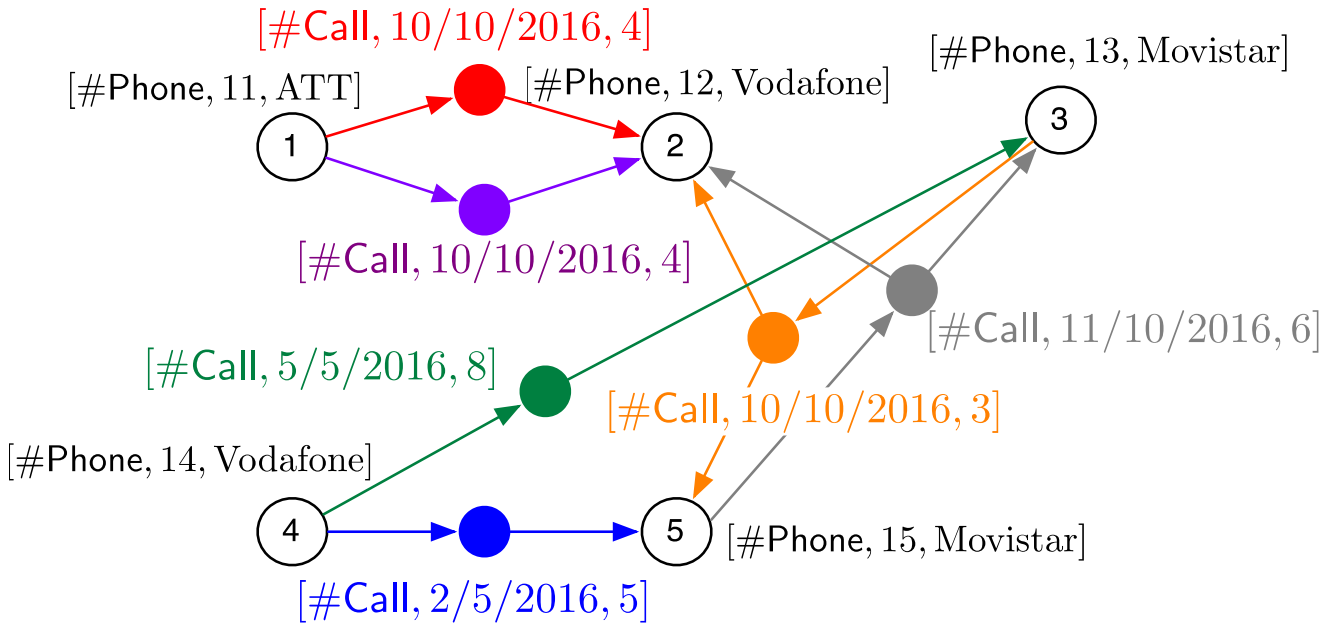
- Facts represented as multi-hypergraphs
- Base Graph (facts)
  - Node types
    - Prefixed by "#"
  - Identifiers
    - Members of the Identifier dimension
  - Attributes
    - Levels in the background dimensions
  - Example
    - [#Phone, 11,$Ph_1$] , [#Phone, 13,$Ph_3$] ,




- Edge types: Same as node types, identifier not mandatory
  - Example
    - [#Call, 2/5/2016,5] , [#Call, 11/10/2016, 6]

# Data model

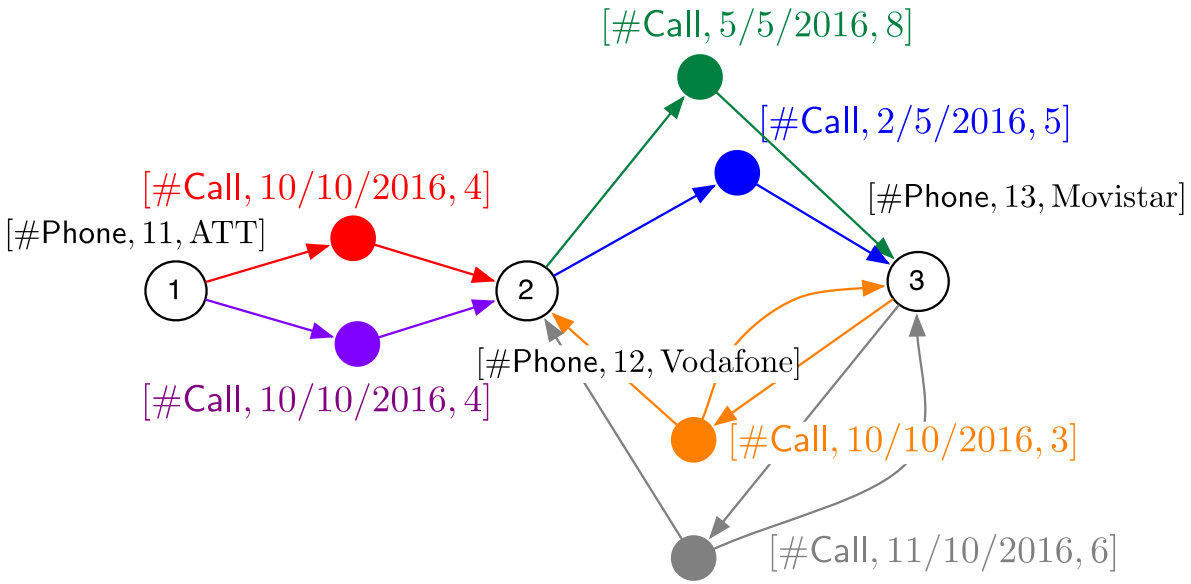
- Graphoids: Graphs at differente granularities (i.e., like cuboids in classic OLAP)
- (Phone.Operator, Time.Day)-Graphoid




- Nodes 12 and 14 correspond to Vodafone, can be merged
- Nodes 13 and 15 correspond to Movistar, can be merged
- An alternative (Phone.Operator, Time.Day)-Graphoid exists

# Data model

- Alternative (minimal) (Phone.Operator, Time.Day)-Graphoid
- Nodes 2 and 4, 3 and 5, are merged




- A minimal graphoid exists and it is unique!

# Outline

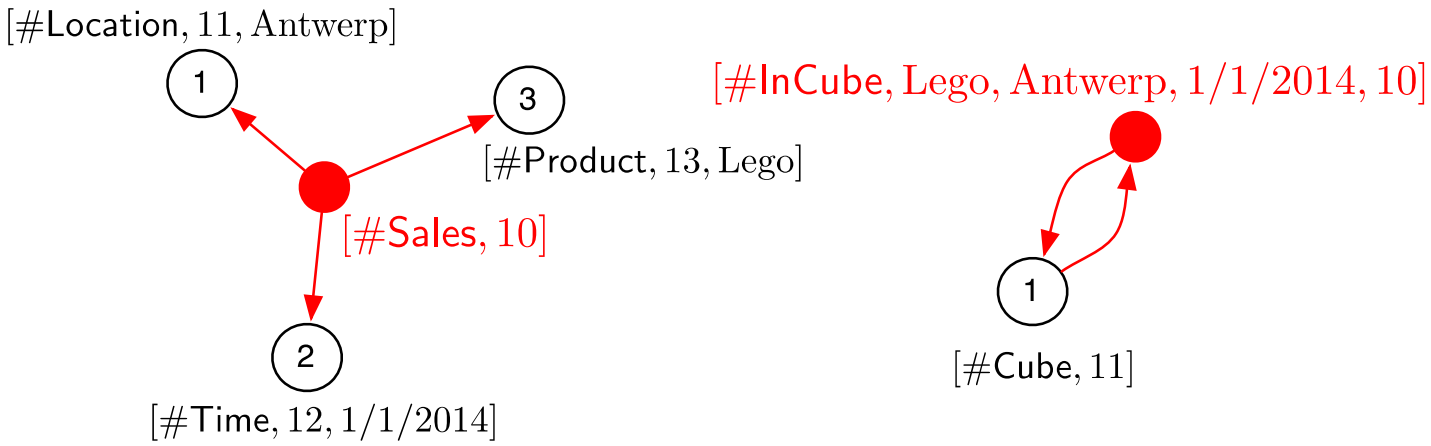
- Introduction and motivation
- Data model
- OLAP over graph data
- Implementation
- Conclusion

# OLAP on graph data

- We want to simulate typical OLAP over graphs
- Operations: Roll-up, Drill-down, Dice, Slice
- Theorem:

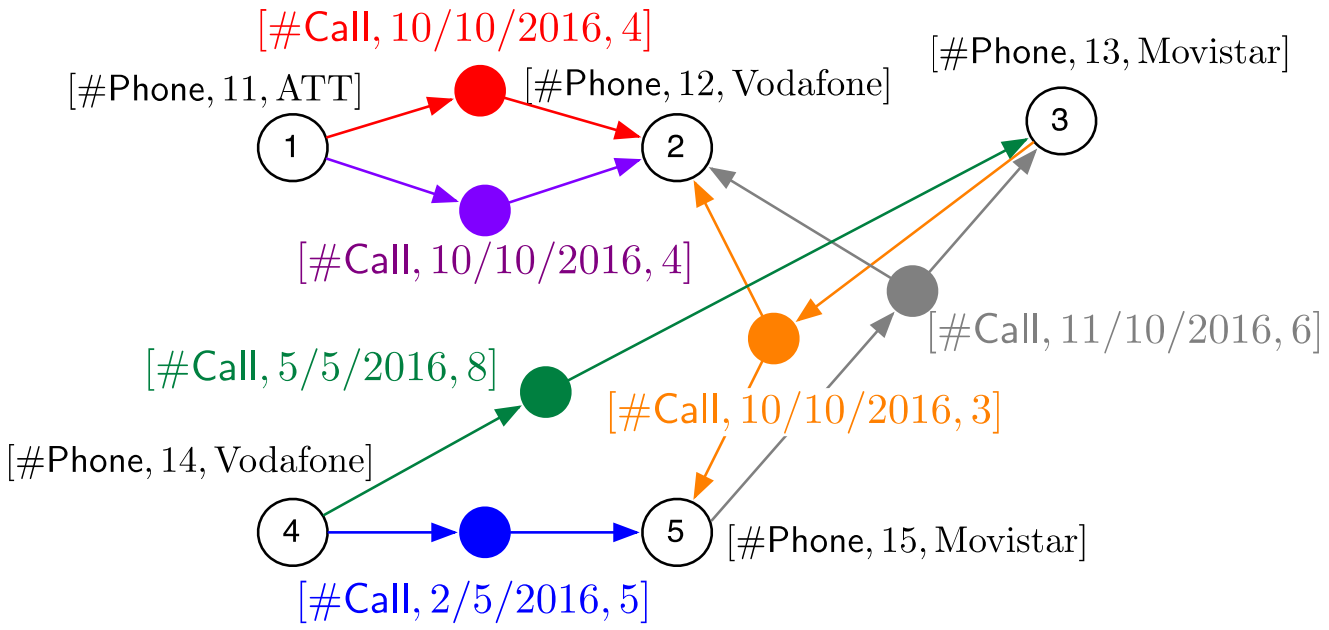
  *The cube OLAP-operations Roll-Up, Drill-Down, Slice and Dice can be expressed (or simulated) by OLAP-operations on graphoids.*

- Typical OLAP is a particular case of graph OLAP, many possible representation alternatives



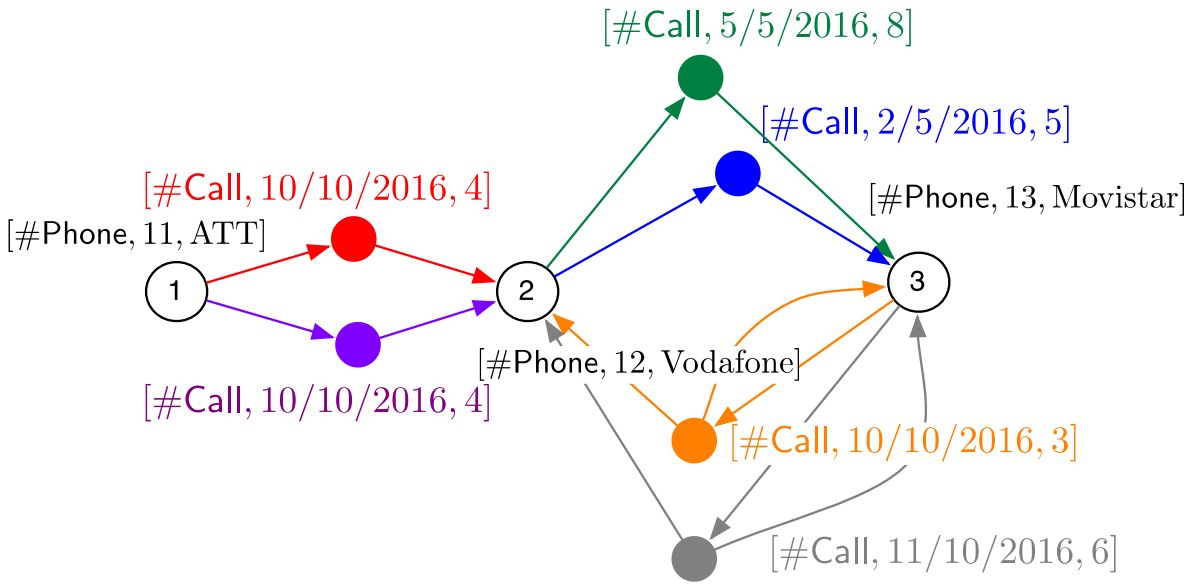
# Operators: Roll-up

- Composition of the operations: Climb, minimization, aggregation
  - Climb(G,# Phone; Phone.(Phone -> Operator))
  - "Climbs" to the "Operator" level, without aggregation (just replaces the level value)
  - Returns the graphoid in previous slide (14),



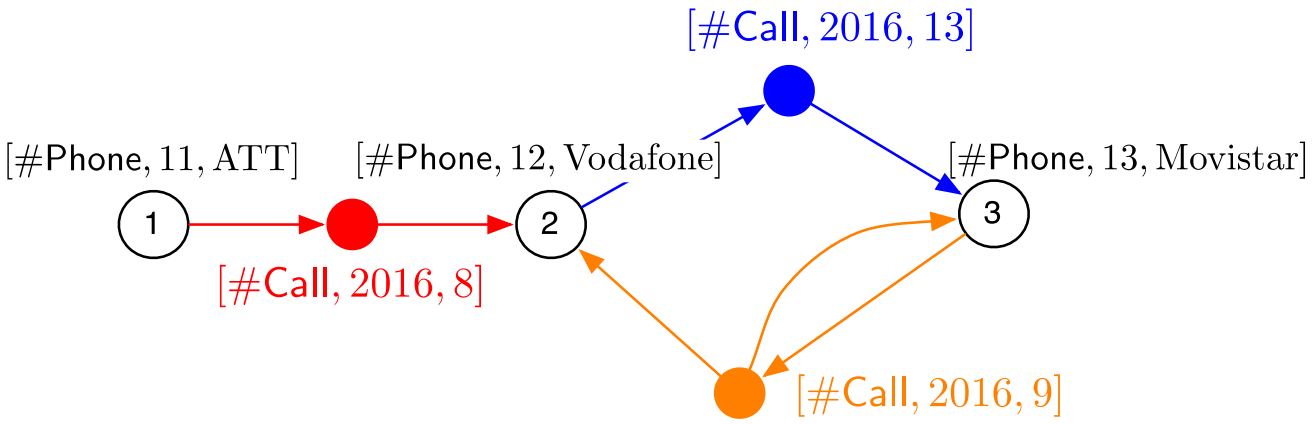
# Operators: Roll-up

- Composition of three operations: Climb, minimization, aggregation
  - Minimize (G;# Phone; Phone.(Phone -> Operator)) *returns the minimal graphoid*
  - *Minimization of the result of Climb*



# Operators: Roll-up

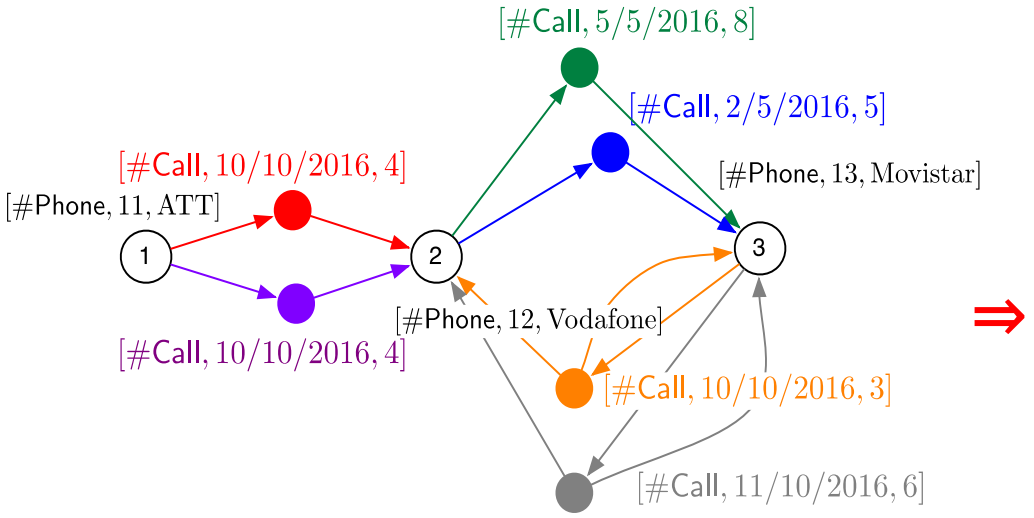
- Composition of three operations: Climb, minimization, aggregation
  - An aggregation is performed over the minimal graphoid
  - *Applied over hyperedges*
  - Roll-Up(G, {# Phone}, Time.(Day -> Year), # Call, Duration, Sum) *over the minimal graphoid in the previous slide, returns*



# Operators: Dice

- Dice(G, $\varphi$) produces a subgraphoid of G, whose nodes are the nodes of G and whose edges satisfy the conditions expressed by $\varphi$

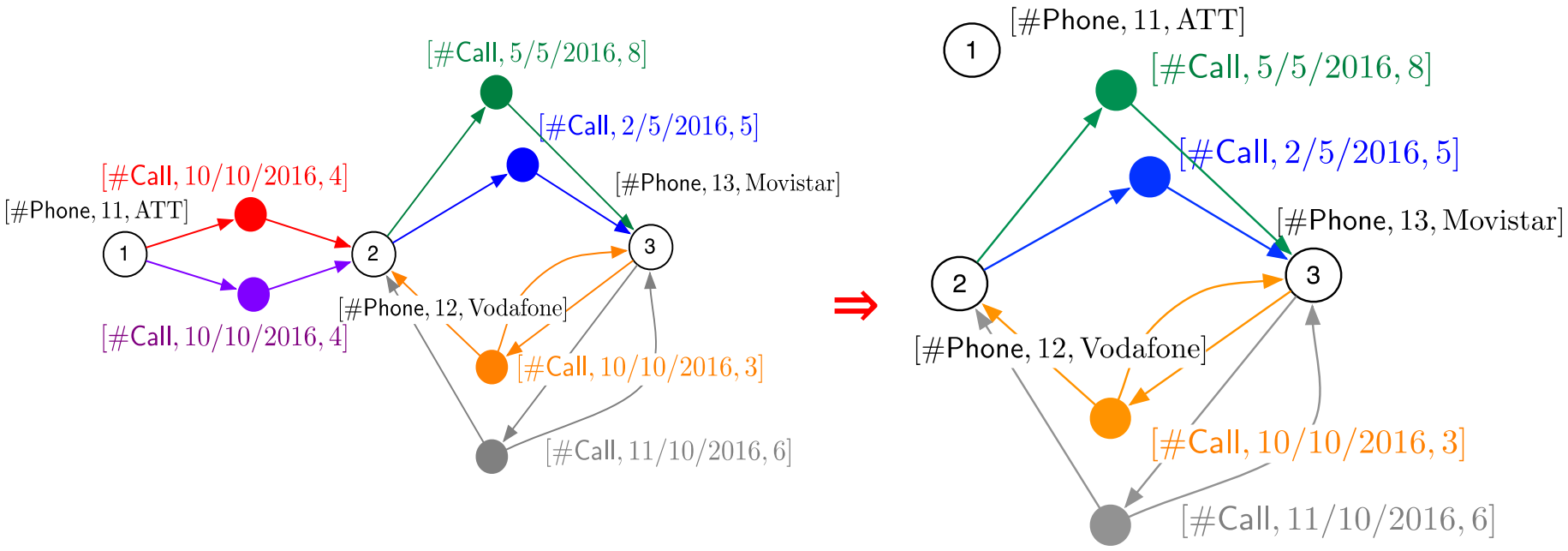
- Dice(G, Phone.Operator != “ATT”) over the graphoid:



# Operators: Dice

- Dice(G, $\varphi$) produces a subgraphoid of G, whose nodes are the nodes of G and whose edges satisfy the conditions expressed $\varphi$

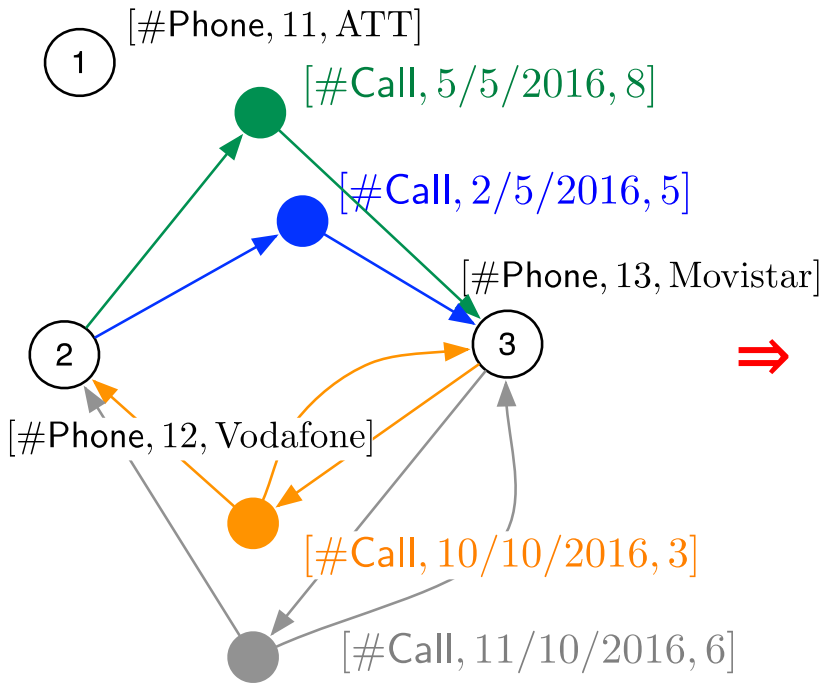
- Dice(G, Phone.Operator != “ATT”)



# Operators: Dice

- Dice(G, $\varphi$) produces a subgraphoid of G, whose nodes are the nodes of G and whose edges satisfy the conditions expressed $\varphi$

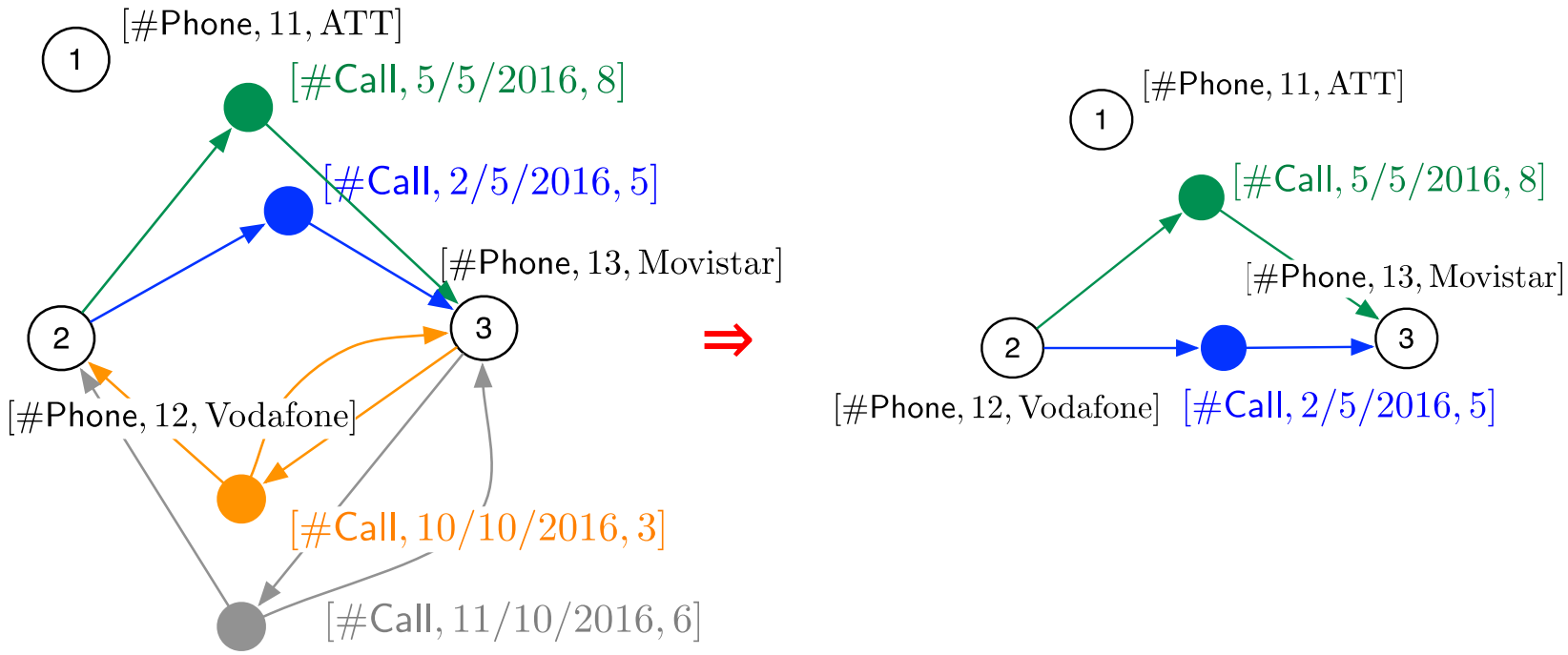
- Dice(G, Time.Month= “5/16”)



# Operators: Dice

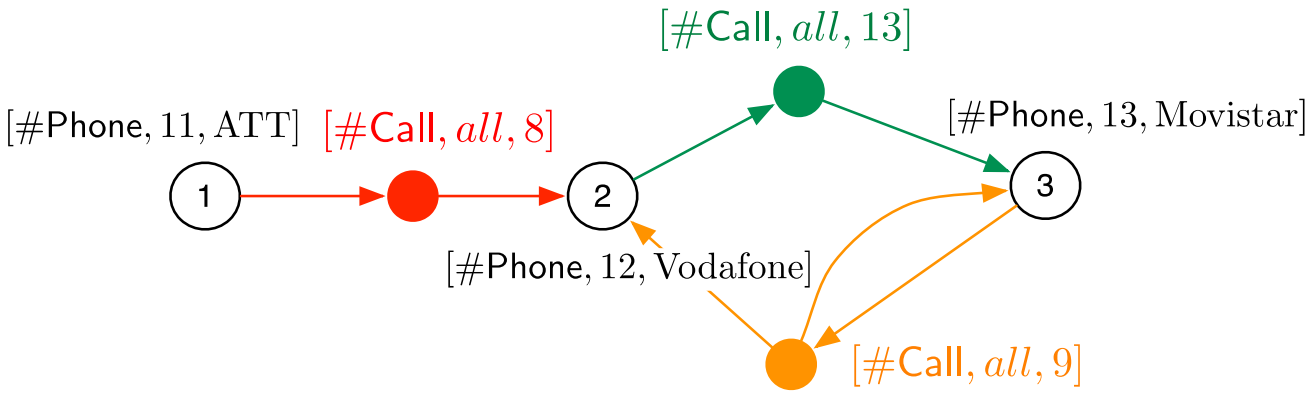
- Dice(G, $\varphi$) produces a subgraphoid of G, whose nodes are the nodes of G and whose edges satisfy the conditions expressed $\varphi$

- Dice(G, Time.Month= “5/16”)



# Operators: Slice

- Slice(G, $D_s$ ; $M_1$ ,…,$M_k$ ,$F_1$ ,…,$F_k$) is the roll-up to the level $D_s$.All over the measures $M_1$ ,…,$M_k$
- Slice(G, **Time**, Duration, Sum)



# Outline

- Introduction and motivation
- Data model
- OLAP over graph data
- Use case and implementation
- Conclusion

# Use Case: IMDB

- Problem: Analyze prizes and nominations for people in the film-making industry.
- Background dimensions
  - *Movie*
    - *Movie -> All*
  - *GeoPerson*
    - *Person -> Country -> All*
  - *Role*
    - *Role -> All*
  - *Time*
    - *Year -> All*
  - *Award*
    - *Award -> Organization -> All*

# Representation in classic OLAP

- Dimensions and facts represented as relations
- One table per dimension
- Two Fact tables – i.e., cubes - (measures are in red)
  - *Participation (Movie, Person, Role, Participates)*
  - *Nomination (Movie, Person, Award,Year, Nominated, Won)*

- Means that for complex OLAP queries, a Drill-Across must be performed
  - That is, fact tables need to be joined

- Queries expressed in Cube Algebra, “representation agnostic” language
  - Queries expressed at conceptual level
  - OLAP queries: sequence of operations

# Representation in classic OLAP

- Query 1: “Number of movies where Woody Allen participated as an actor”.
- In Cube Algebra:

```
Q1 <- Dice(Participation, Person = 'Woody Allen')
Q2 <- Dice(Q1, Role='Actor')
Q3 <- Slice(Q2, Role, count)
Q4 <- Slice(Q3, Movie, count)
```

# Representation in classic OLAP

- Query 2: “Pairs of Movies and Persons, such that only people who played more than one role in it (other than Director) participated, listing only persons who were nominated for an Oscar in that movie, but did not win the award”.
- In Cube Algebra:

Q9 <- Dice(Participacion, Role <> 'Director')

Q10 <- Slice(Q9, Role, SUM)

Q11 <- Dice(Q10, Participates > 1 )

Q12 <- RollUp(Nomination, Award -> Organization,SUM,SUM)

Q13 <- Dice(Q12, Award.Organization = 'Oscar Award’ AND Won=0)

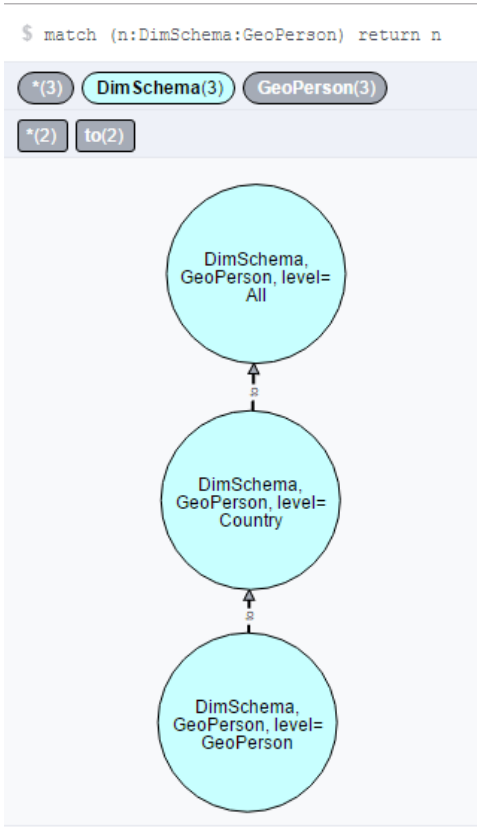
Q14 <- Slice(Q13, Award, SUM, SUM)

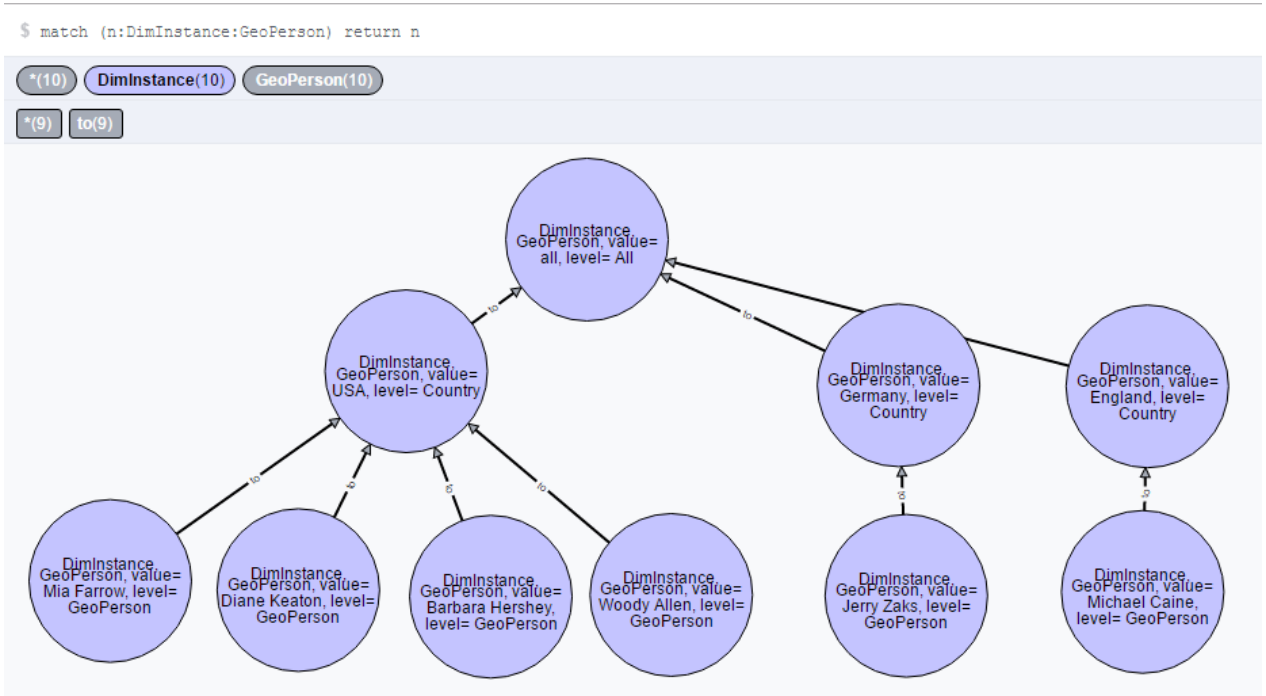
Q15 <- Slice(Q14, Year, SUM, SUM )

Q16 <- DrillAcross(Q11, Q15)

# Representation as graphs

- Implementation in Neo4j (dimensions)



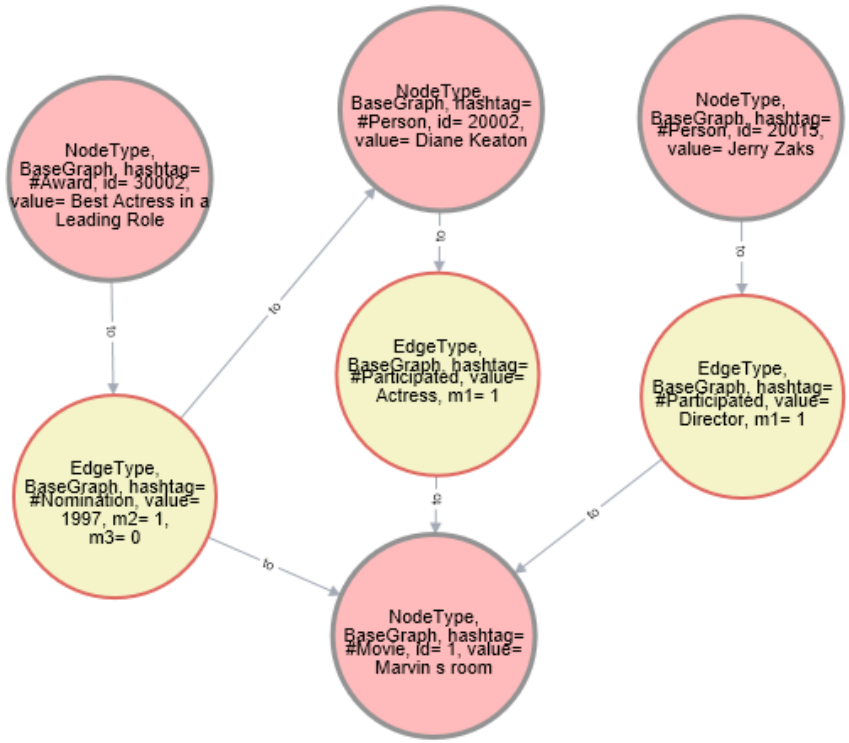



# Representation as graphs

- Base Graph and Graphoids
  - Node and edge types represented as nodes
  - Two edge types in the example
    - *#Participated*
    - *#Nomination*
  - Three node types
    - *#Movie*
    - *#Person*
    - *#Award*
  - Role and Time are represented as attributes in the edge type nodes

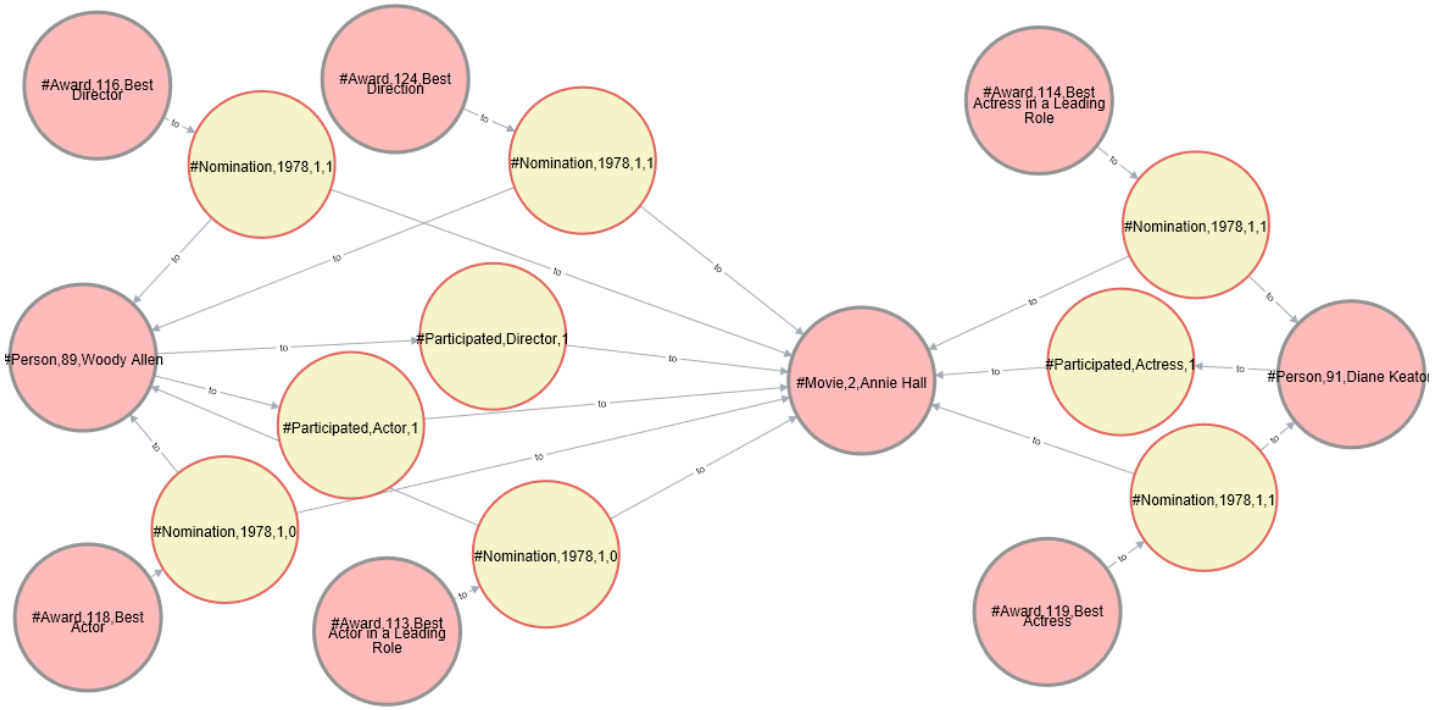
# Representation as graphs

- *Implementation in Neo4j (Base Graph)*



# Representation as graphs

- *Base Graph in Neo4j*



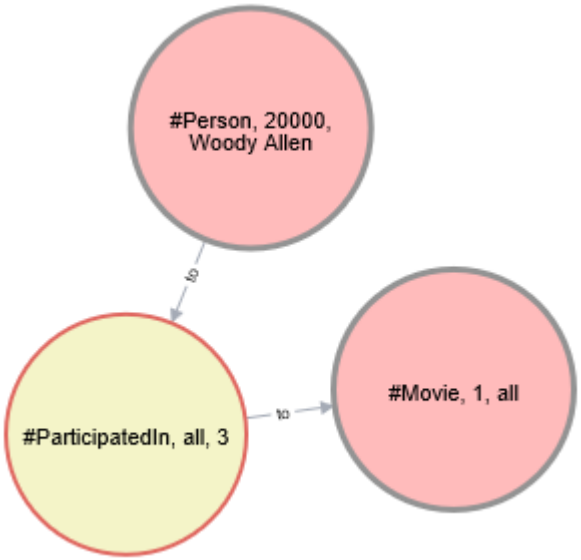
# Representation as graphs

- Query 1: “Number of movies where Woody Allen participated as an actor”.
- In Graph OLAP:

```
Operators.Dice(graphDB,"BaseGraph","Q1","GeoPerson",
                    "GeoPerson", "=", "Woody Allen" );
Operators.Dice(graphDB,"Q1","Q2","Role","Role","=”, "Actor");
Operators.Slice(graphDB, "Q2","Q3","Role”, Arrays.asList("m1","m2","m3"),
                    Arrays.asList(COUNT,COUNT,COUNT));
Operators.Slice(graphDB,"Q3","Q4","Movie”, Arrays.asList("m1","m2","m3"),
                    Arrays.asList(COUNT,COUNT,COUNT));
```

# Representation as graphs

- *Query 1*



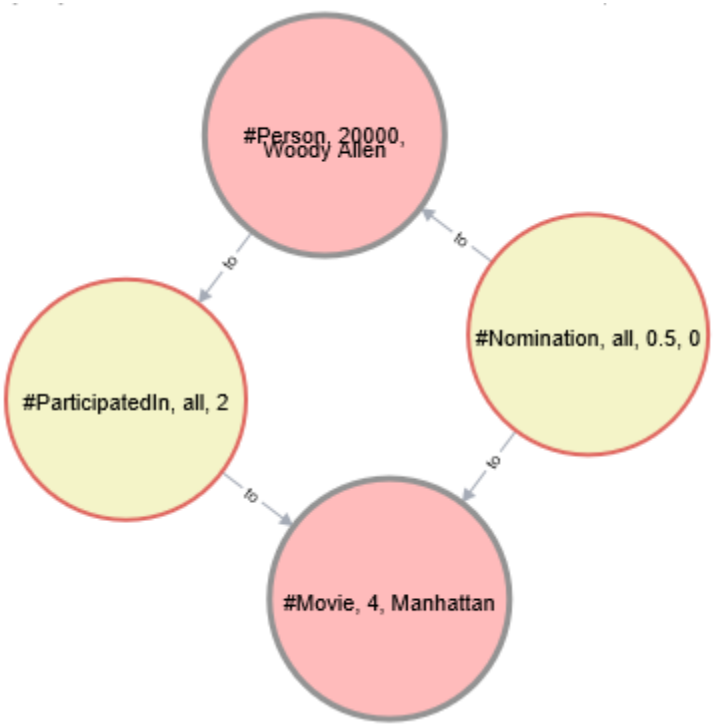
# Representation as graphs

- Query 2: “Pairs of Movies and Persons, such that only people who played more than one role in it (other than Director) participated, listing only persons who were nominated for an Oscar in that movie, but did not win the award”.
- In graph OLAP:

```
Operators.Dice(graphDB,"BaseGraph","Q9","Role","Role”, "<>", "Director");
Operators.Slice(graphDB,"Q9","Q10","Role”, Arrays.asList("m1","m2","m3"),  Arrays.asList(SUM,SUM,SUM));
Operators.Dice(graphDB,"Q10","Q11","measures","m1”, "<>","1.0");
Operators.Rollup(graphDB,"Q11","Q12”, Arrays.asList("#Award"), "Award", "Award",
        "Organization","#Nomination”, Arrays.asList("m1","m2","m3"),Arrays.asList(SUM,SUM,SUM));
Operators.Dice(graphDB,"Q12","Q13","Award”, "Organization","=", "Oscar Award" );
Operators.Dice(graphDB,"Q13","Q14","measures”, "m3","=","0");
Operators.Slice(graphDB, "Q14","Q15","Award”, Arrays.asList("m1","m2","m3"), Arrays.asList(SUM,SUM,SUM));
Operators.Slice(graphDB, "Q15","Q16","Year”, Arrays.asList("m1","m2","m3"), Arrays.asList(SUM,SUM,SUM));
```

# Representation as graphs

- *Query 2*



# Some preliminary results

- Neo4J to store graphs
- No code optimization
- Ran individual operations for different graph sizes
  - Dice
  - Slice
  - Rollup
- We used portions of the IMDB data set
- HW: i7-6700 processor, 12 GB RAM, 250GB disk (a virtual node in a cluster)

# Some preliminary results

| | # movies | # nodes | # edges | Size (MB) | DICE 1 sec | DICE 2 sec | SLICE sec | ROLLUP 1 sec | ROLLUP 2 sec |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 320 | 8576 | 14208 | 7 | 0.33 | 0.65 | 55 | 56.3 (16.7) | 52.1 (15.7) |
| 2 | 640 | 17152 | 28416 | 13 | 0.5 | 0.87 | 220 | 195 (50.7) | 206 (51.7) |
| 3 | 1280 | 34304 | 56832 | 25 | 0.6 | 1.3 | 740 | 769 (188) | 728 (174) |
| 4 | 2560 | 68608 | 113664 | 50 | 1.2 | 2.1 | 3010 | 2977 (672) | 2972 (679) |

DICE 1: Dice (G, Role = 'Actor')

DICE 2: Dice (G, Role <> 'Director')

SLICE: Slice (G, Role, SUM, SUM, SUM)

ROLLUP 1: Rollup (G, Award -> Organization, SUM, SUM, SUM)

ROLLUP 2: Rollup (G, Award -> All, SUM, SUM, SUM)

- Between parentheses: Climb + Minimize
- No cloning considered
  - We measured processing time + time to store the result on disc

# Discussion

- Reasonable results for small/medium graphs
  - No problem with DICE
  - *Minimize* operation does not harm
  - Need to work on the aggregation part for Rollup/Slice
  - Cloning is not an option with Neo4j. Too slow
    - Existing proposals use materialize views even for single operations
  - For larger graphs, or complex queries, parallelization is mandatory
- However….
  - More flexible than ROLAP/MOLAP
  - Allows much expressive queries (e.g., aggregation + Shortest paths or other measure over graphs, not possible in classic OLAP)
  - For the use cases presented, the relational option would also be expensive
    - In the IMDB example, joining + aggregating Fact Tables is expensive
    - The Phone Calls example may imply computing the transitive closure of the Fact Table

# Outline

- Introduction and motivation
- Data model
- OLAP over graph data
- Use case and implementation
- Conclusion

# Conclusion

- Intention of this work
  - State and study the problem
  - Propose a data model
  - Present a proof-of-concept implementation based on Neo4J
- Many things remain to be addressed
  - More (compelling) case studies
  - Query processing
    - Better algorithms for aggregation
    - Neo4j not the best option, cloning prohibitive
    - Parallelism needed, materialize graphoids *á la* Pagrol?

Thank you!