Extending In-Memory Relational Database Engines with Native Graph Support

EDBT'18

Mohamed S. Hassan[1] Tatiana Kuznetsova[1] Hyun Chai Jeong[1]
Walid G. Aref[1] Mohammad Sadoghi[2]

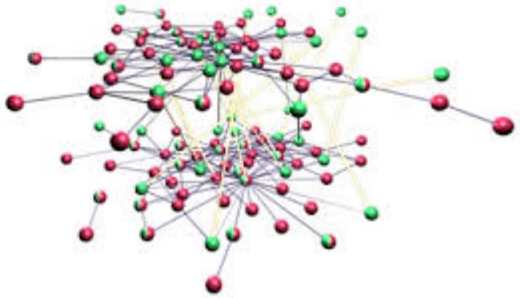
[1]Purdue University – West Lafayette, IN, USA

[2]Exploratory Systems Lab (ExpoLab)
[2]University of California – Davis, CA, USA

Graphs are Ubiquitous

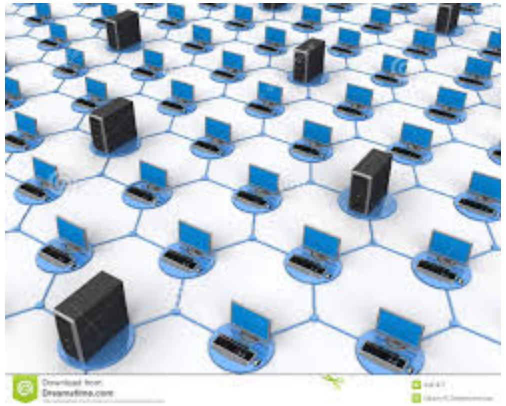
Road Network

Biological Network



Social Network

Datacenter Network



Specialized Graph Databases

- Specialized graph databases can handle graph query-workloads
  - Vital queries include **shortest-path** and **reachability** queries

Why Relational Databases for Graph Support?

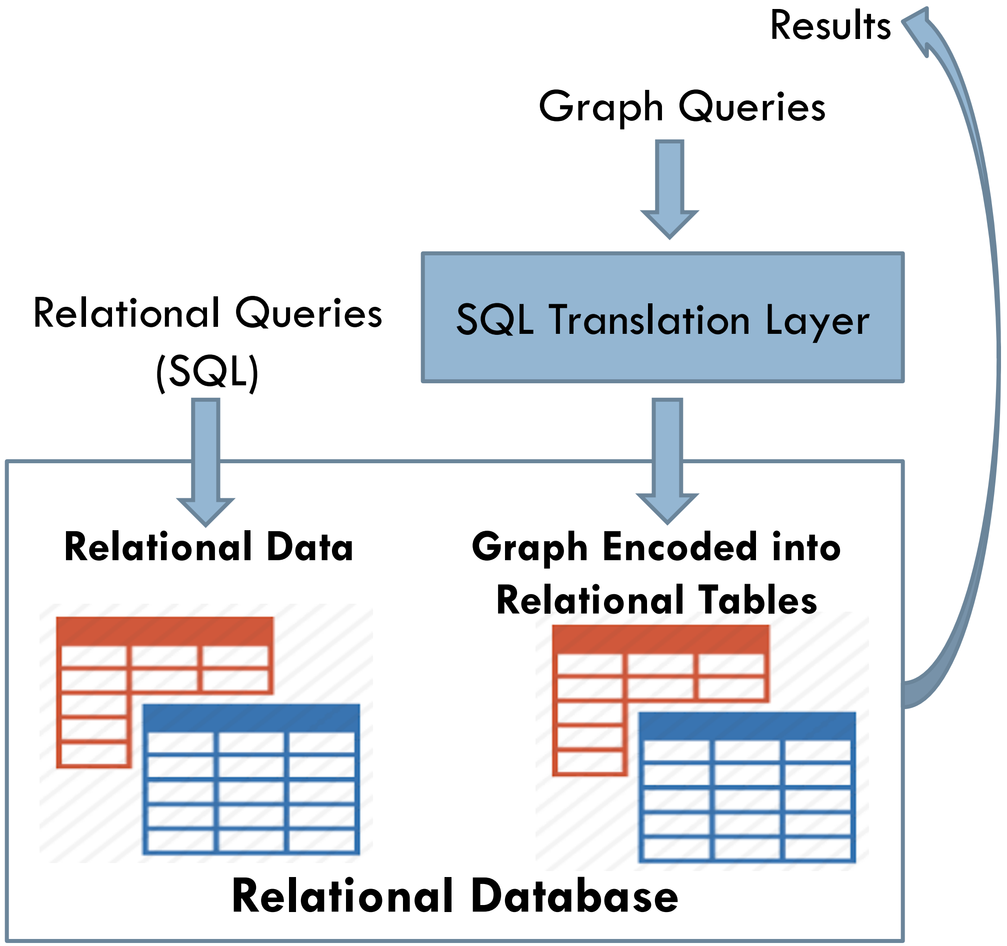
- Specialized graph systems are not as mature as RDBMSs
  - Relational databases are widely-adopted
- Graphs and RDBMSs
  - Relational data can have latent graph structures
  - Graphs can be represented in terms of relational tables
- Graph queries are essential in many applications
  - Queries can also involve relations
    - E.g., for every patient, say P, in selected areas, find the **nearest** hospital to Patient P
- How can an RDBMS effectively and efficiently handle graph query workloads?

Graph Support in RDBMSs

- Why is it challenging?
  - There is an impedance mismatch between the relational model and the graph model
- Graph support w.r.t. RDBMSs has two extremes:
  - Native Relational-Core
  - Native Graph-Core
  - **Native G+R Core [Proposed]**

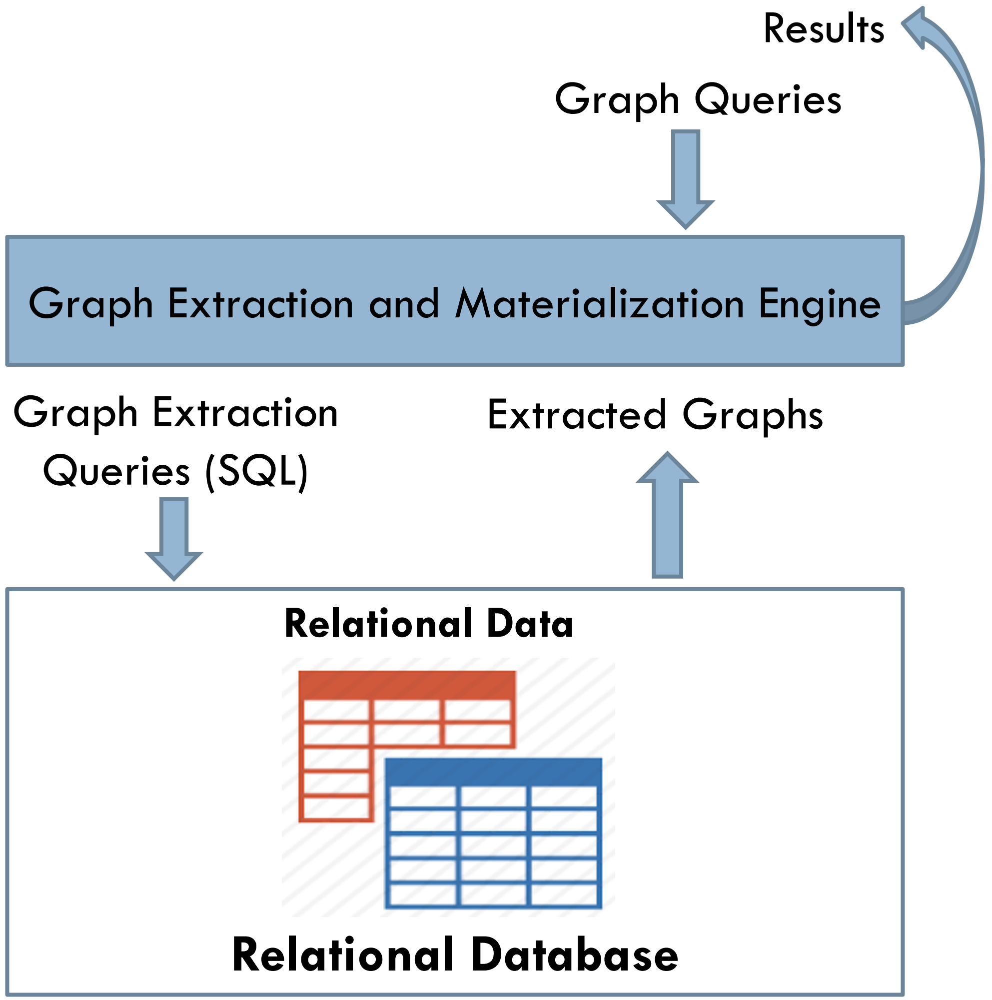
Native Relational-Core

- Use a vanilla RDBM
- Encode graphs in relational schema
- Support limited graph queries
- Translate the supported graph queries into SQL or procedural SQL
- E.g., SQLGraph [SIGMOD'15], Grail [CIDR'15]
- Disadvantages
  - Several graph queries are inefficient to evaluate using pure SQL
  - Graphs are encoded in complex schema



Native Graph-Core

- Build on top of an RDBMS
- Extract graphs from the RDBMS
- Store graphs and process queries outside the realm of the RDBMS
- E.g., Ringo [SIGMOD'15], GraphGen [VLDB'15, SIGMOD'17]
- Disadvantages
  - Graph updates require re-extracting the graphs
  - Queries cannot reference any non-extracted relational data

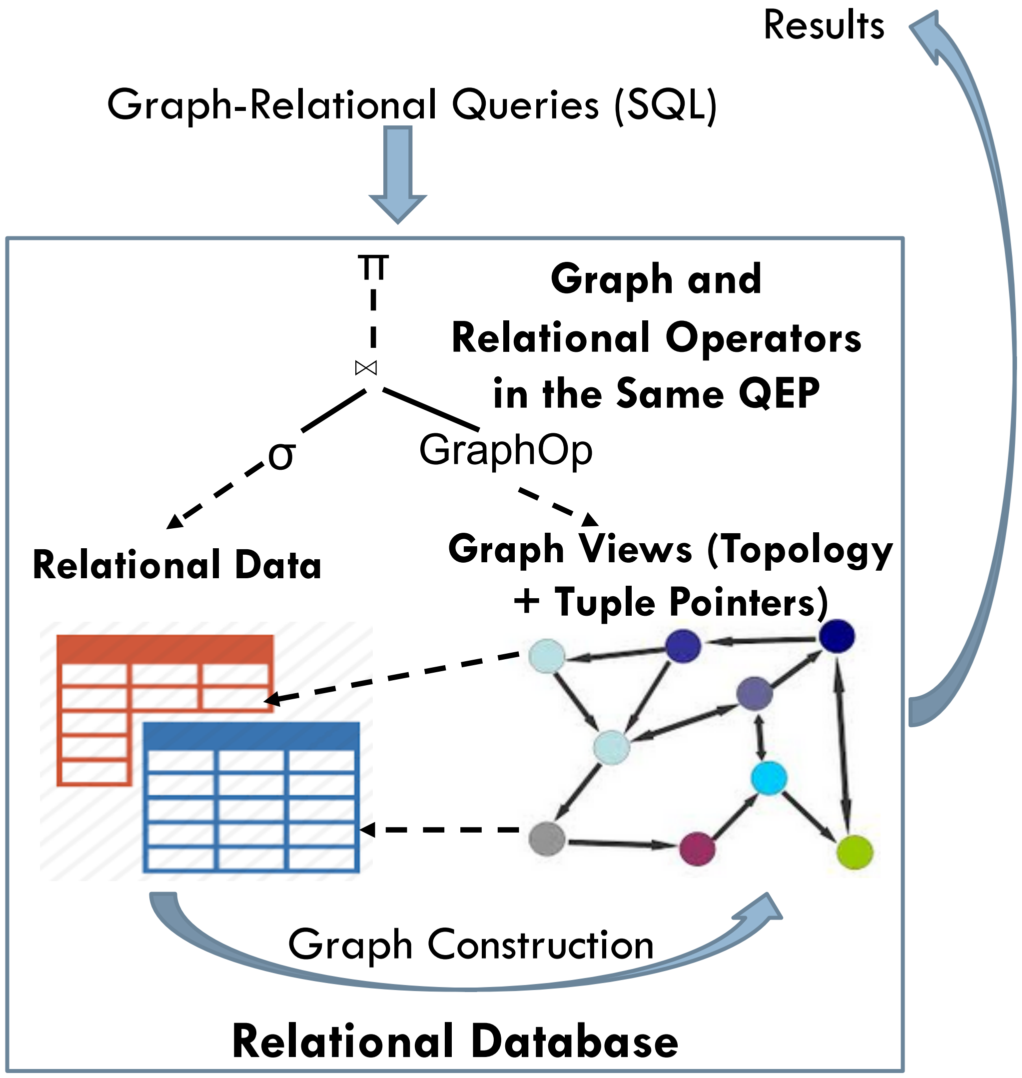


The Relational Model vs. the Graph Model

- Graph-core approach
  - +ve: Queries involving graph traversals are efficiently handled in the graph model (e.g., shortest paths)
  - -ve: Not as pervasive and mature as RDBMSs
- Relational-core approach
  - +ve: Mature and pervasive
  - -ve: Either many temporary inserts/deletes/updates, or too many joins to traverse a graph
    - Intermediate-result size and cardinality estimation
- Can the best of the two worlds be combined?
  - Support native graph processing inside an RDBMS

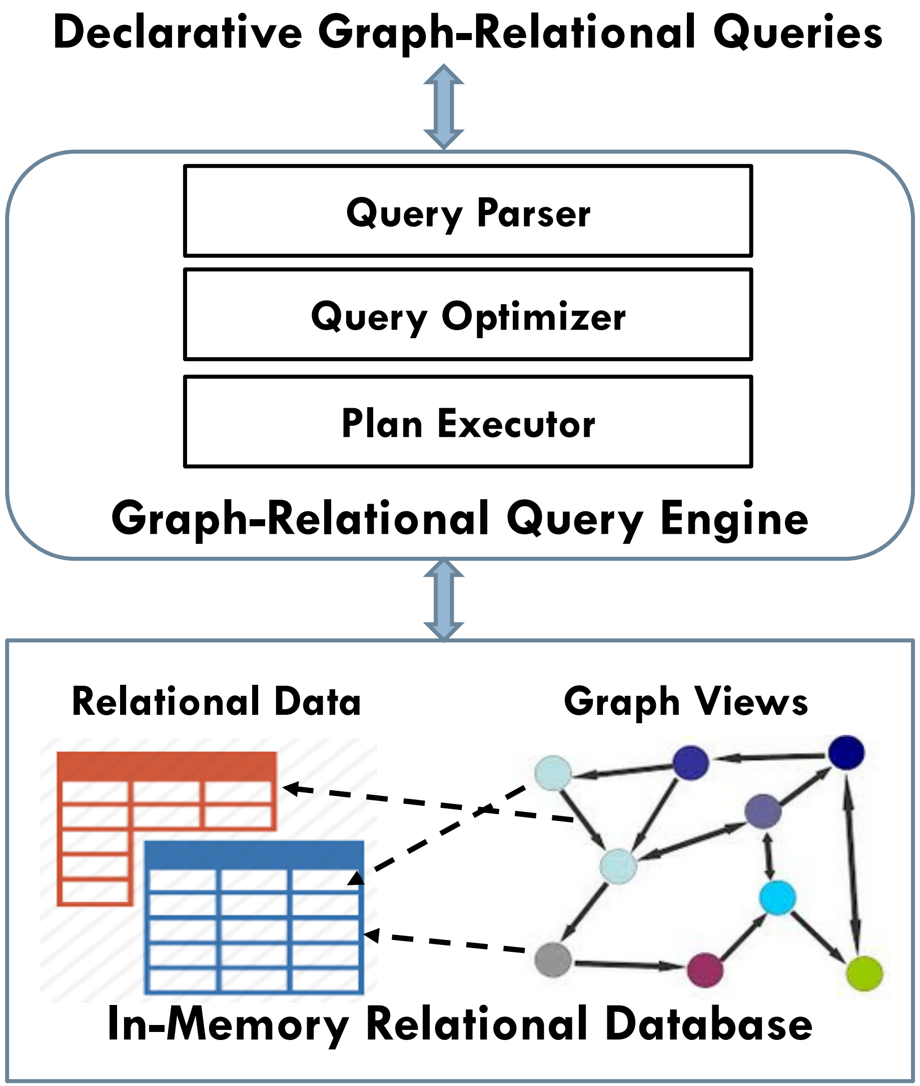
Proposed Approach: Native G+R Core

- Assume graphs with relational schema
- Enable graphs to be defined as native database objects
- Store graphs in non-relational structures optimized for graph operations
- Extend the SQL language
  - Queries can compose relational and graph operations
- Cross-Data-Model QEPs
- Graph updates are supported



GRFusion: Realizing the G+R Approach

- We realized the G+R approach in an open-source in-memory RDBMS, VoltDB
  - We refer to the realization as **GRFusion**



Create Graph View

- Create-Graph-View statement
  - Creates a named graph database object that can be referenced in queries
  - Defines the relational sources of the graph's vertexes/edges
  - Martializes the topology of the graph in the main-memory as a singleton graph structure

```
CREATE [DIRECTED | UNDIRECTED] GRAPH VIEW
    ↪ gview_name
VERTEXES ( ID = idCol [, { vPropName =
    ↪ vPropCol } [ ,...n ] ] )
FROM relational_src1 [where_clause1]
EDGES ( ID = idCol, FROM = srcVId, TO =
    ↪ destSID [, { ePropName = ePropCol } [
    ↪  ,...n ] ] )
FROM relational_src2 [where_clause2]
```

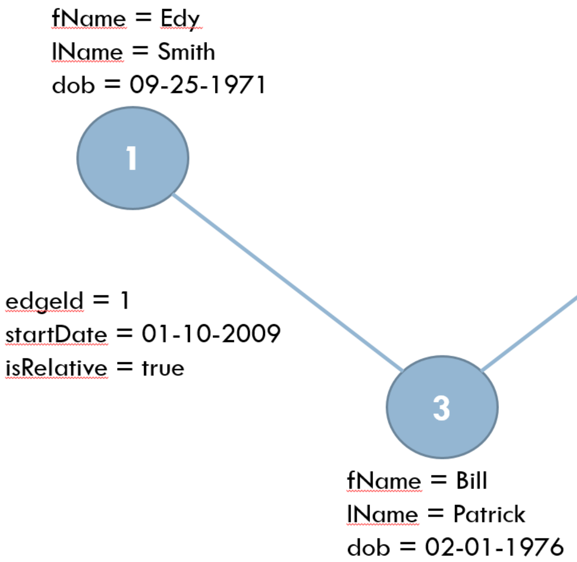
Graph-View of a Social Network

Users

uId	fName	lName	dob
1	Edy	Smith	09-25-1971
2	Jones	Parker	11-21-1980
3	Bill	Patrick	02-01-1976
…..	…..	……	……

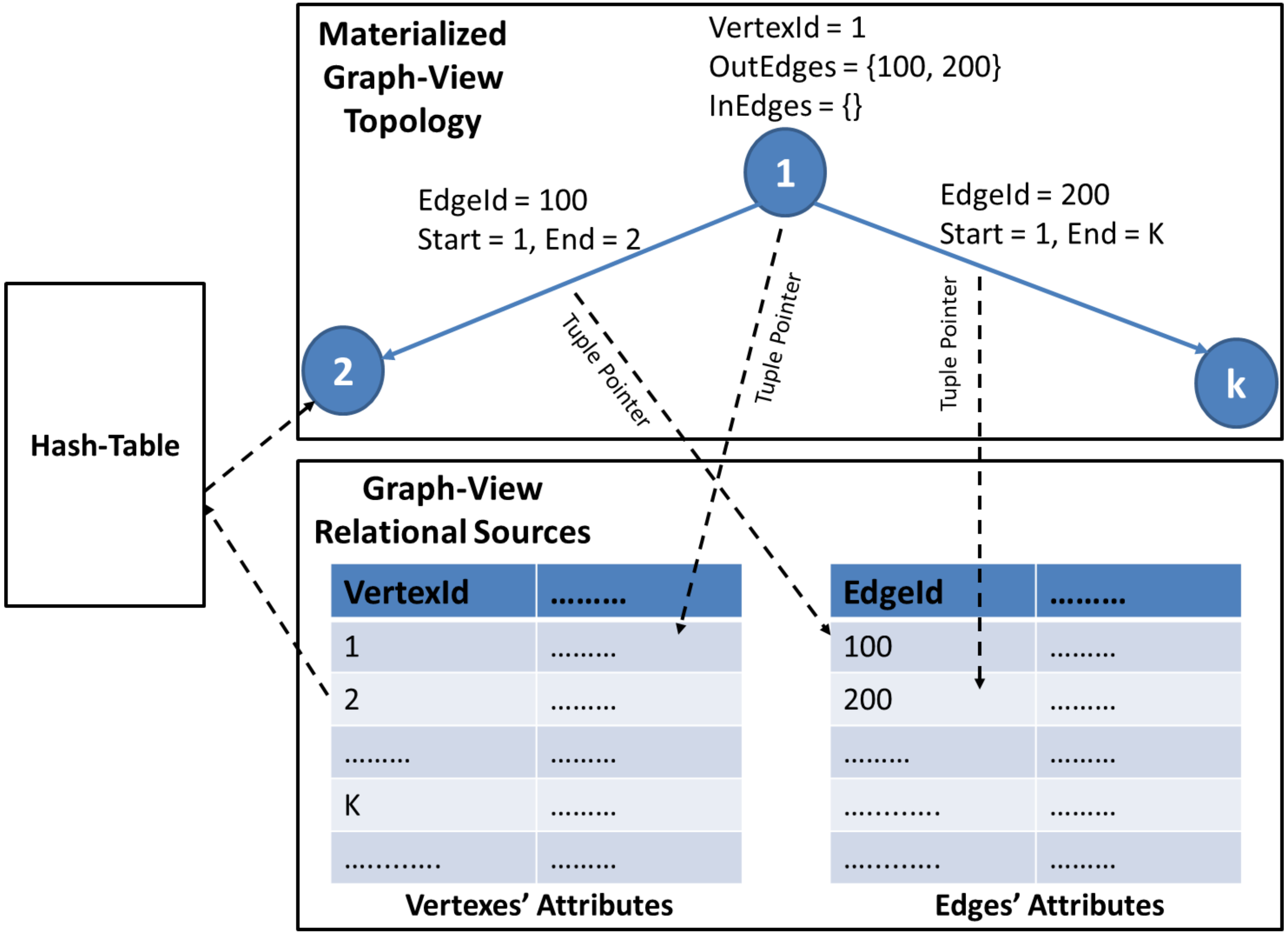
Relationships

relId	uId1	uId2	startDate	isRelative
1	1	3	01-10-2009	true
2	2	3	12-31-2008	false
	…..	…..	……	……




```
CREATE UNDIRECTED GRAPH VIEW SocialNetwork
VERTEXES(ID = uId, lstName = lName,
    ↪ birthdate = dob)
FROM Users
EDGES (ID = relId, FROM = uId1, TO = uId2,
    ↪ startDate = sDate, relative =
    ↪ isRelative)
FROM Relationships
```

Graph-View Structure [Traversal Index]



Declarative Graph-Relational Queries


```
SELECT <select_list>
[ FROM { relational_src | graph_vw |
    ↪ graph_vw_vertexes | graph_vw_edges |
    ↪ graph_vw_paths } [ ,...n ] ]
[ WHERE <search_condition> ]
```

The PATHS Construct – Extended SQL

- Appears in the FROM clause and references a graph view
  - **Select … From MyGraphView.PATHS** P
- PATHS represents a set of lazy-evaluated paths
- A path is a set of consecutive edges, each edge has two endpoint vertexes
  - E.g., (V:attributes) **–(:E:attributes)→**(V:attributes) …..
- A path is a tuple with the following properties:
  - Length
  - StartVertex
  - EndVertex
  - Vertexes
  - Edges

The PathScan Operator

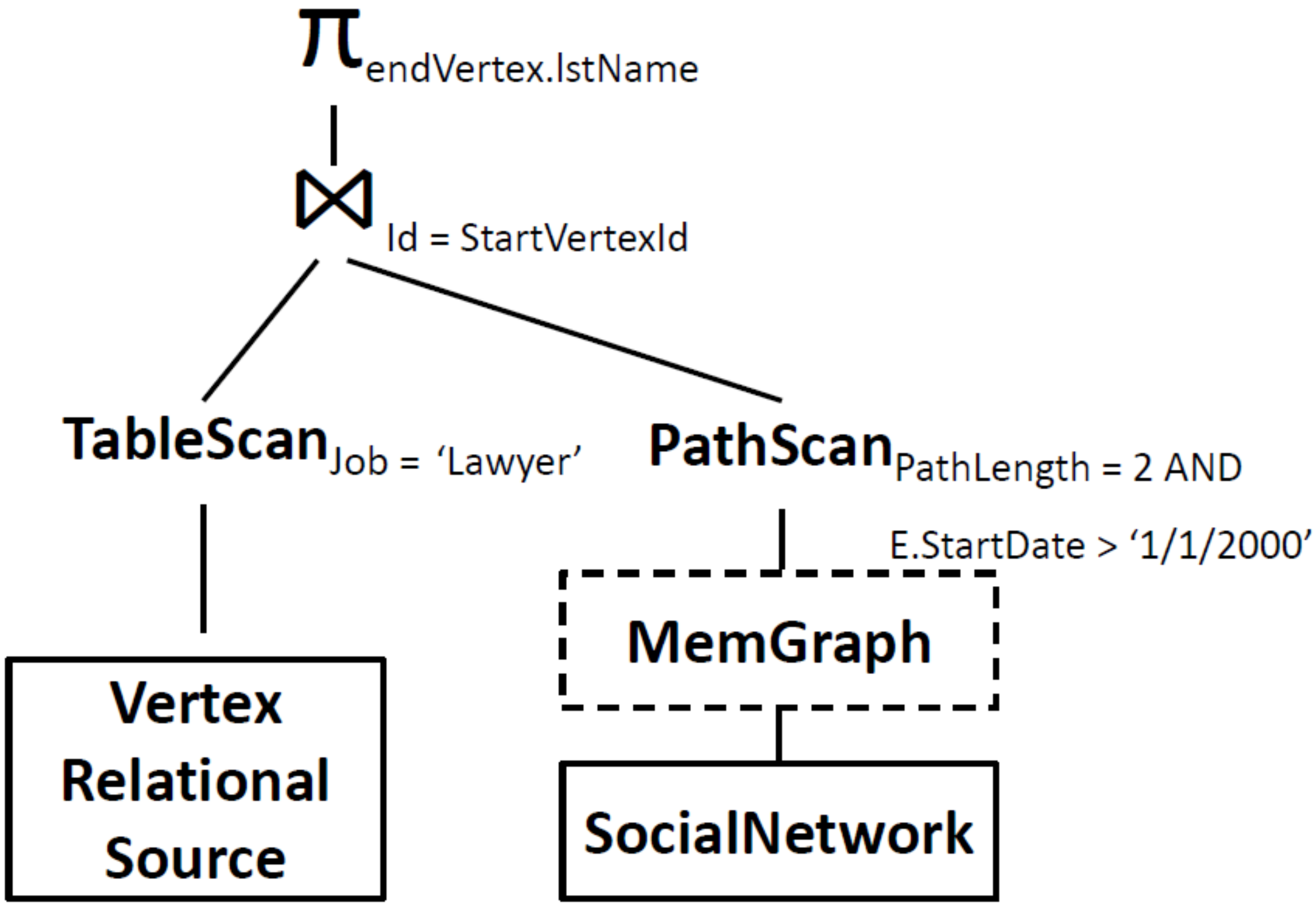
- PathScan is a logical operator that acts on a graph-view
  - Has three corresponding physical operators: BFScan, DFScan, SPScan
- The output of PathScan is a tuple that extends the standard relational tuple
  - Hence, the output can be ingested by any relational operator
- PathScan accepts the id of the vertex to start traversal from
  - Otherwise, all the vertexes will be considered as start vertexes
- Filters can be pushed ahead of PathScan operators
  - E.g., P.PathLength = 2

Friends-of-Friends Query Example

- For all the users working as lawyers, retrieve the last name of their friends of friends, where the friendships happened after 1/1/2000

```
SELECT PS.EndVertex.lstName
FROM Users U, SocialNetwork.Paths PS
WHERE U.Job = 'Lawyer' AND PS.StartVertex.Id
    ↪  = U.uId AND PS.Length = 2 AND PS.
    ↪ Edges[0..*].StartDate > '1/1/2000'
```

QEP of the Friends-of-Friends Query



Reachability Query Example

- Check if Protein X interacts directly (i.e., by an edge) or indirectly (i.e., by a path) with Protein Y through either a covalent or a stable interaction type.

```
SELECT PS.PathString
FROM Proteins Pr1, Proteins Pr2, BioNetwork.
    ↪ Paths PS
WHERE Pr1.Name = 'Protein X' AND Pr2.Name =
    ↪ 'Protein Y' AND PS.StartVertex.Id =
    ↪ Pr1.Id AND PS.EndVertex.Id = Pr2.Id
    ↪ AND PS.Edges[0..*].Type IN ('covalent
    ↪ ', 'stable')
LIMIT 1
```

Shortest-Path Queries with Relational Predicates


```
SELECT TOP 2 PS
FROM RoadNetwork.Paths PS HINT(SHORTESTPATH(
    ↪ Distance)), RoadNetwork.Vertexes Src,
    ↪  RoadNetwork.Vertexes Dest
WHERE PS.StartVertex.Id = Src.Id AND PS.
    ↪ EndVertex.Id = Dest.Id AND Src.
    ↪ Address = "Address 1" AND Dest.
    ↪ Address = "Address 2"
```

Evaluating GRFusion

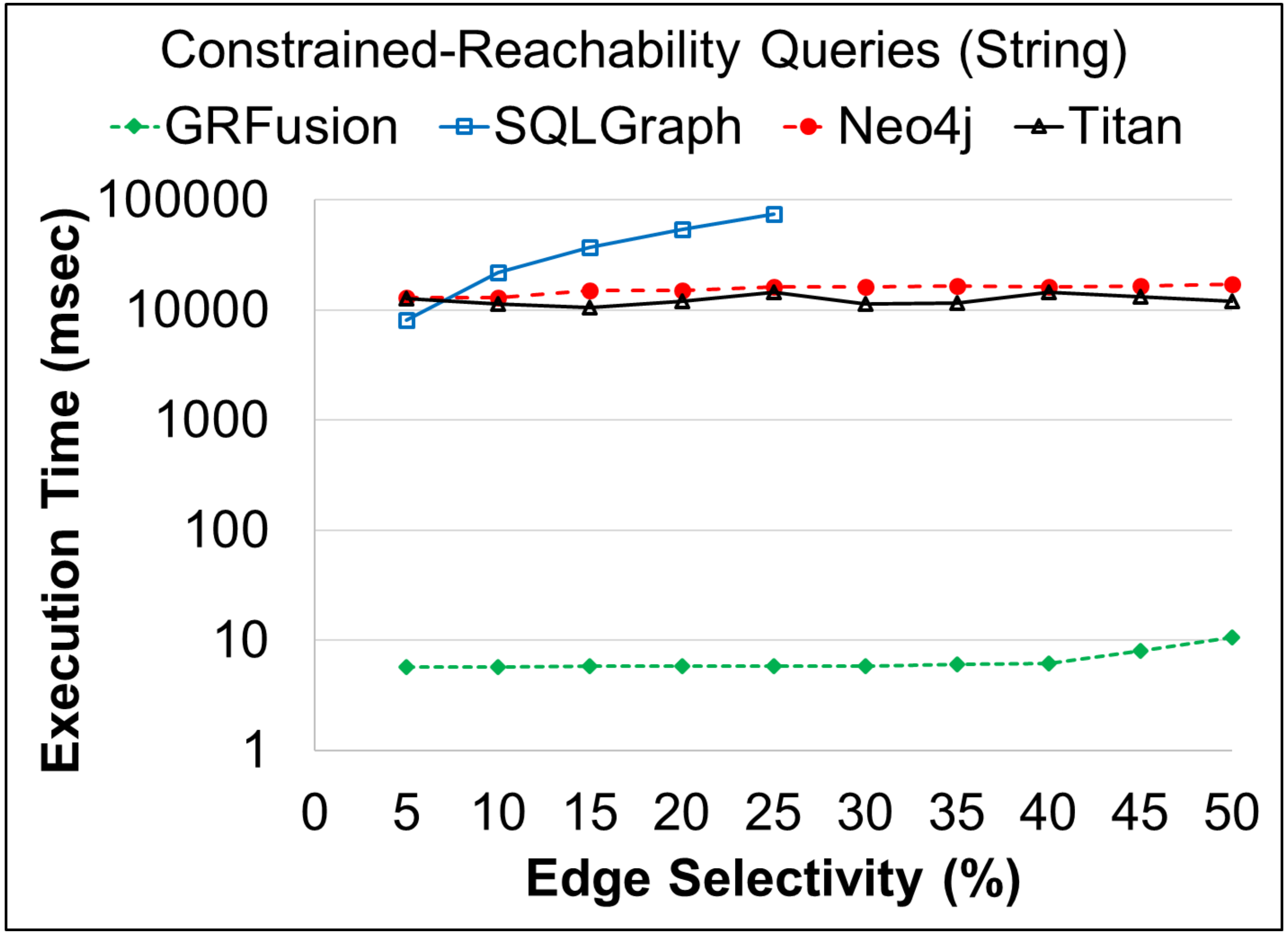
- Experimental setup
  - Single node running Linux kernel version 3.17.7
    - 32 cores of Intel Xeon 2.90 GHz
    - 384 GB of RAM
  - VoltDB version 6.7
- Comparing to
  - Native Relational-Core: SQLGraph [SIGMOD'15], Grail [CIDR'15]
  - Specialized graph systems: Neo4j, Titan
  - Disk-cost is mitigated by running over ram disk

Evaluating GRFusion (Cont'd)

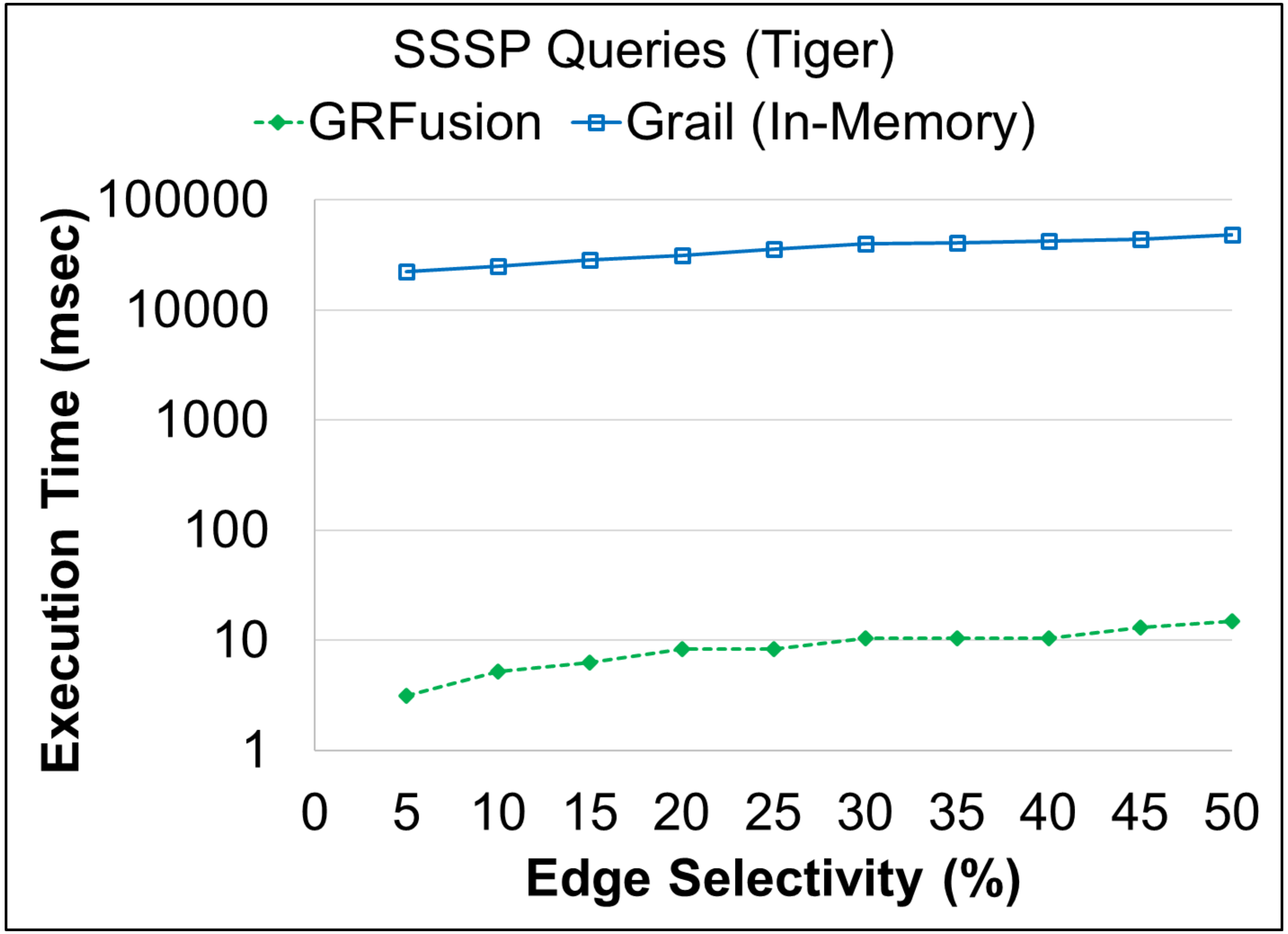
- Graph queries
  - Reachability queries (using breadth-first-search)
  - Reachability queries with filtering predicates
  - Shortest path queries (using Dijkstra's algorithm)
  - Subgraph queries (e.g., count triangles)
- Datasets

Dataset	Number of Vertexes	Number of Edges
Tiger Road Network	24,412,259	58,698,439
DBLP Co-Author Network	1,007,047	6,592,656
String Protein Network	1,520,673	348,473,440
Twitter Follower Network	41,652,230	1,468,365,182

Constrained-Reachability Queries (String Dataset)

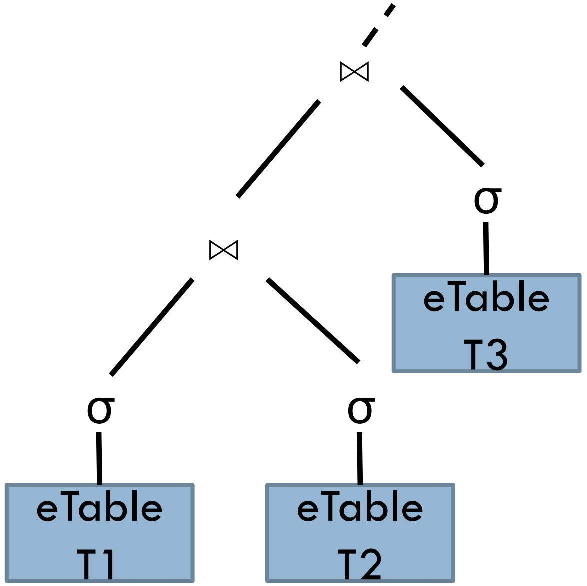


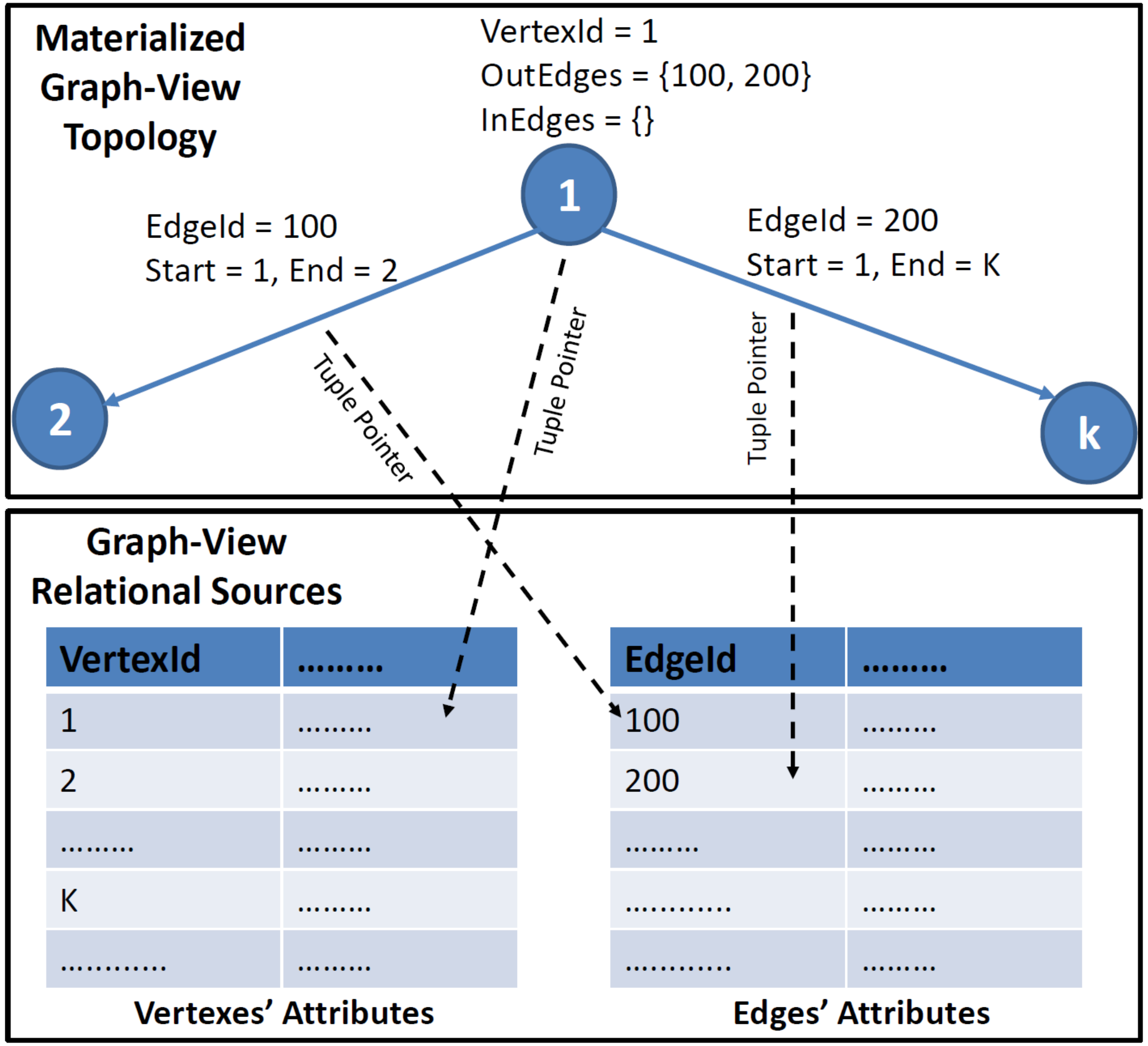
SSSP Queries – Tiger Dataset



A Note on the Performance Gains of GRFusion

- Table scan or index scan/seek
  - Direct pointers are more efficient
- Relational joins
  - Large intermediate results
  - Inaccurate cardinality estimation





Conclusions

- The G+R approach allows composing relational and graph operations
  - E.g., by allowing graph-valued functions
- GRFusion proposes and realizes how an RDBMS can be extended to support graphs as native objects
- GRFusion outperforms the state-of-the-art by one to four orders-of-magnitude query-time speedup
- The SQL language of GRFusion allows writing declarative path-queries with relational predicates
- For relational recursive queries, GRFusion allows an RDBMS to avoid
  - Large intermediate results
  - Inaccurate cardinality estimation that may lead to non-optimal join-algorithm selection

Thank You!

The VERTEXES Construct

- Appears in the FROM clause and references a graph view
  - **Select … From MyGraphView.VERTEXES** v
- VERTEXES represents the vertexes of a graph view
- A vertex is a tuple with the following properties:
  - Id
  - FanIn
  - FanOut
  - Property for each vertex attribute

The EDGES Construct

- Appears in the FROM clause and references a graph view
  - **Select … From MyGraphView.EDGES** v
- EDGES represents the edges of a graph view
- An edge is a tuple with the following properties:
  - Id
  - StartVertexId
  - EndVertexId
  - Property for each edge attribute

Vertex Query Example

- Retrige the Birthdate and the number of friends of each user in the social network with last name = ‘Smith’

```
SELECT VS.birthdate, VS.fanOut
FROM SocialNetwork.Vertexes VS
WHERE VS.lstName = 'Smith'
```