Graph Analytics using Vertica Relational Database

Alekh Jindal* Samuel Madden Malú Castellanos Meichun Hsu

Microsoft MIT Vertica Vertica

* work done while at MIT

Motivation for graphs on DB

- Data anyways in a DB
  - avoid expensive copying
  - end-to-end data analysis
  - leverage other DB features

- Processing involves full scans and joins
  - relational engines could run them efficiently
  - particularly suited for column stores

- Relational algebra/SQL offers powerful declarative syntax
  - in fact, we could express Giraph as an operator DAG
  - can even express more complex graph analytics

5-point Agenda

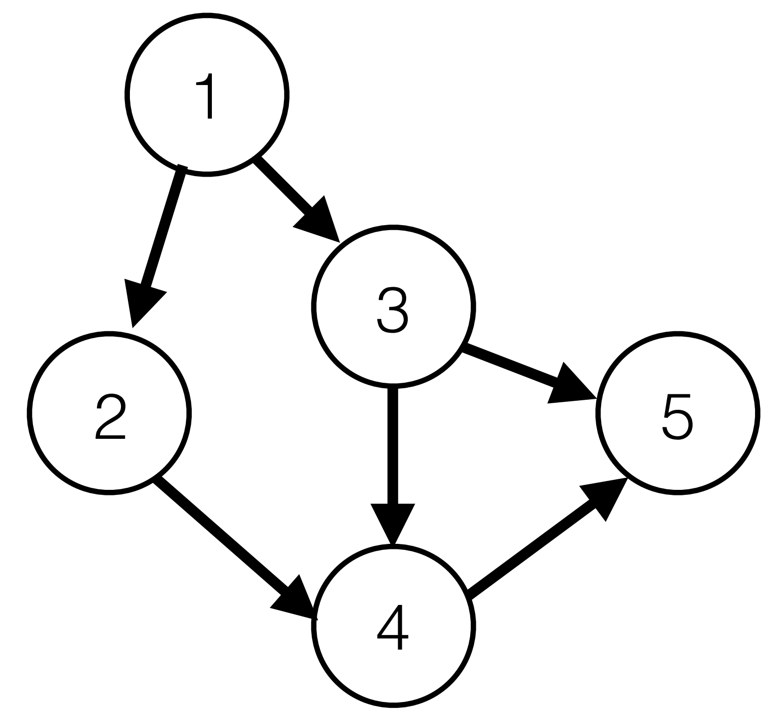
- **From graph queries to SQL**: how do we make the translation?
- **Graph query optimization**: can we leverage decades of relational wisdom?
- **Column store backends**: why are they a good choice?
- **Comparison with specialized graph systems**: how do the numbers look?
- **Extending column stores**: can we do better?

1. From Graph to SQL

Vertex-centric Graph Queries

- Popular language for graph analytics

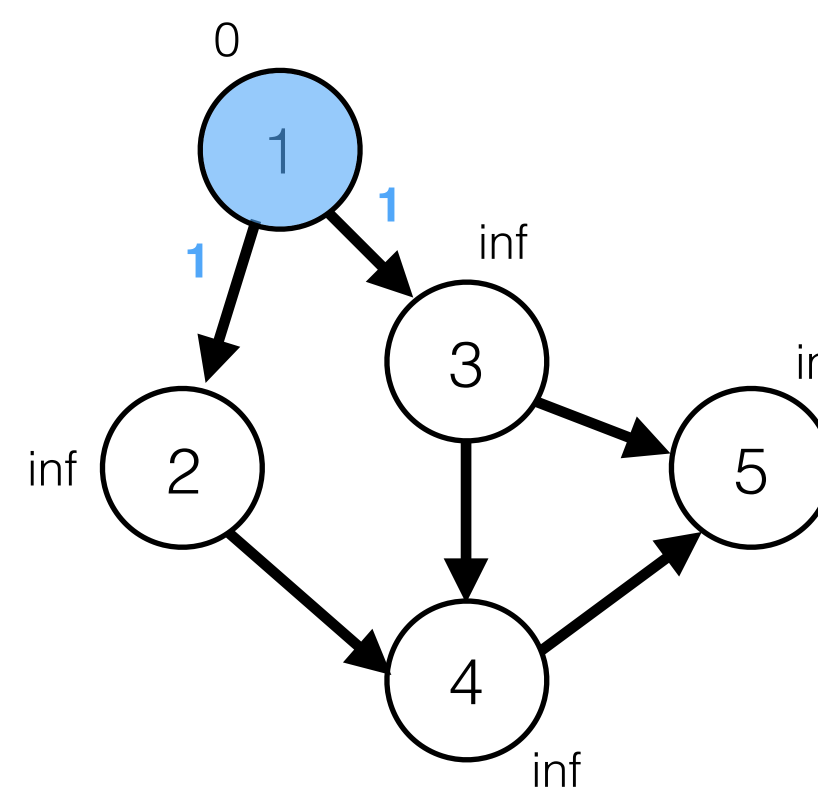
- Vertex programs that run in supersets and communicate via message passing



Vertex-centric Graph Queries

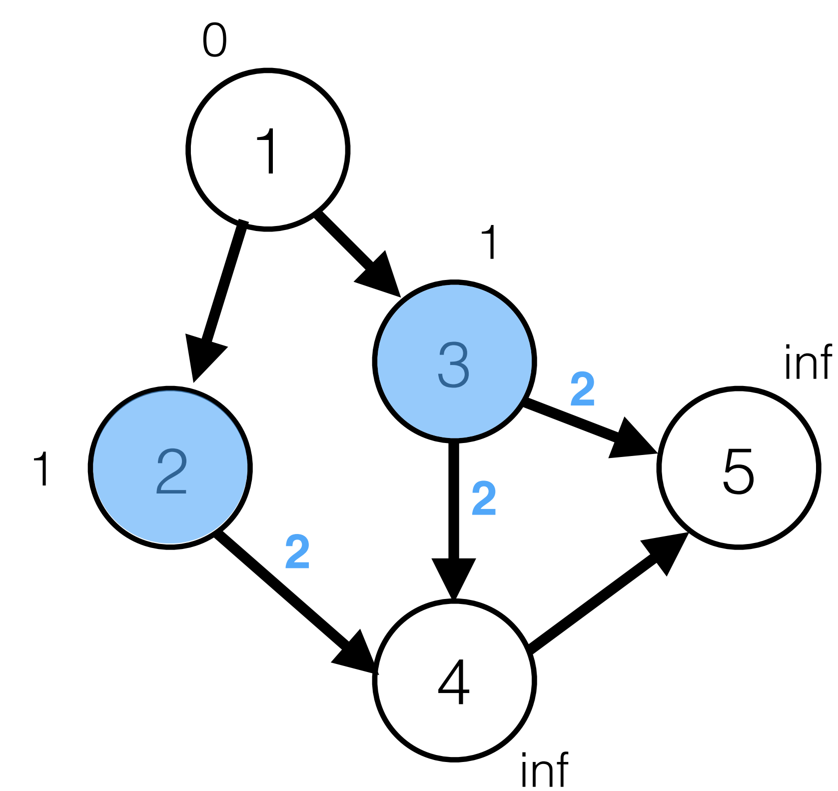
- Popular language for graph analytics

- Vertex programs that run in supersets and communicate via message passing



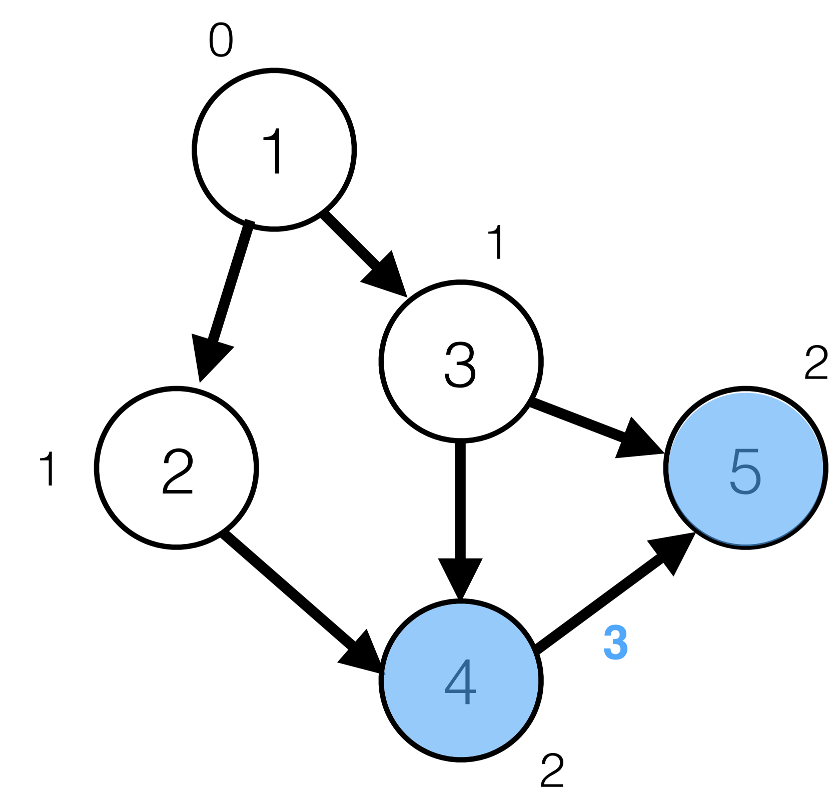
Vertex-centric Graph Queries

- Popular language for graph analytics
- Vertex programs that run in supersets and communicate via message passing



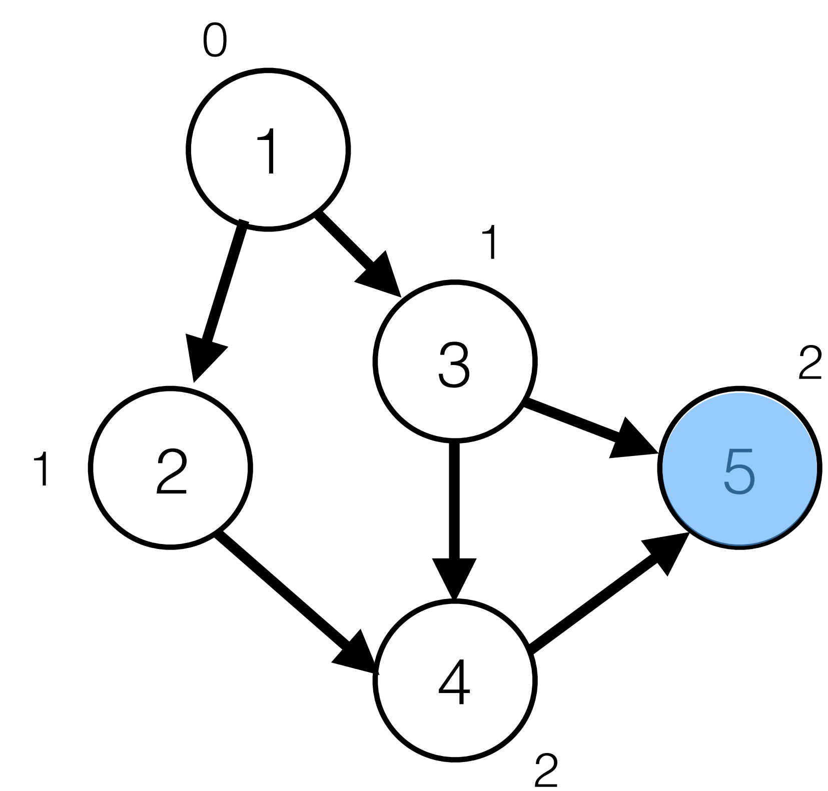
Vertex-centric Graph Queries

- Popular language for graph analytics
- Vertex programs that run in supersets and communicate via message passing



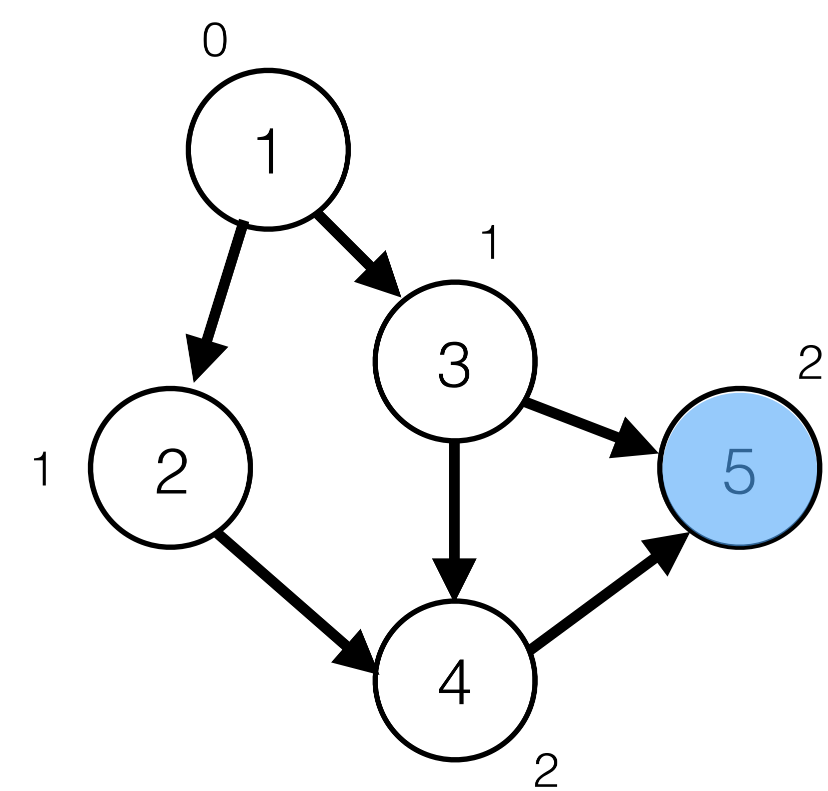
Vertex-centric Graph Queries

- Popular language for graph analytics
- Vertex programs that run in supersets and communicate via message passing



Vertex-centric Graph Queries

- Popular language for graph analytics
- Vertex programs that run in supersets and communicate via message passing
- Programmer only specifies a vertex program
- System takes care of running it in parallel

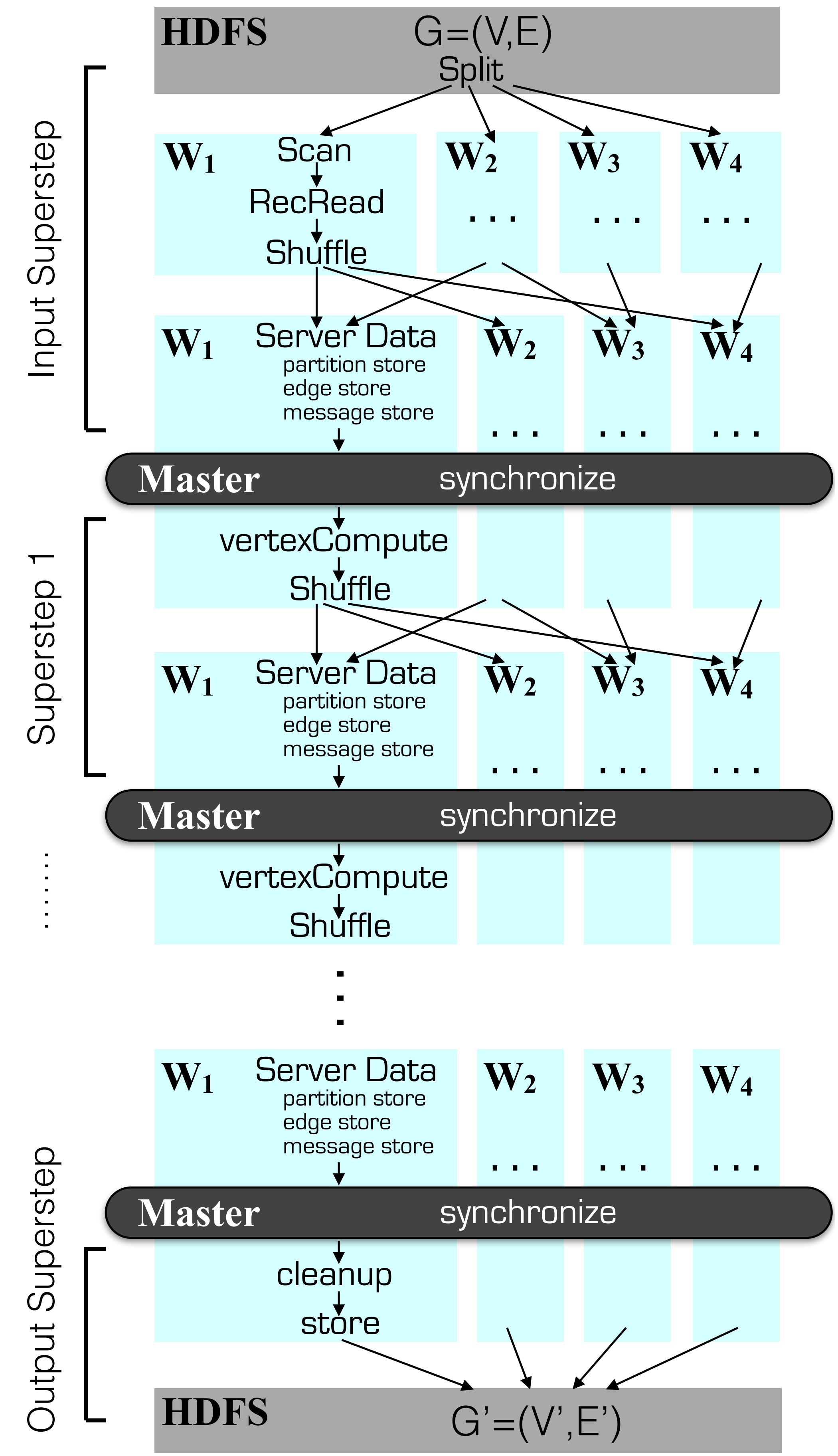


The Giraph Plan

- Giraph: a popular, open-source graph analytics system on Hadoop

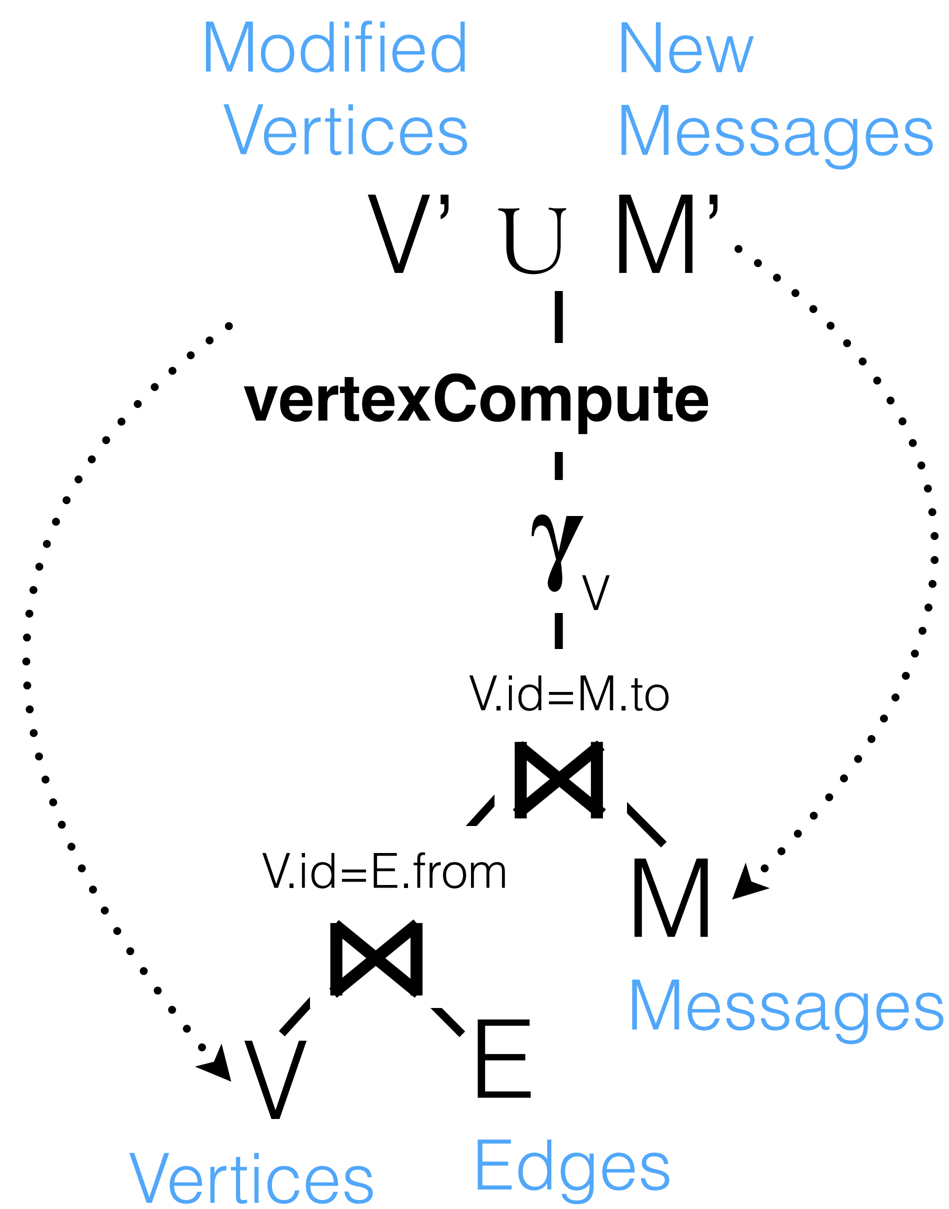
The Giraph Plan

- Giraph: a popular, open-source graph analytics system on Hadoop
- The Giraph physical plan: hard coded physical execution pipeline



The Giraph Plan

- Giraph: a popular, open-source graph analytics system on Hadoop
- The Giraph physical plan: hard coded physical execution pipeline
- Giraph logical query plan using relational operators

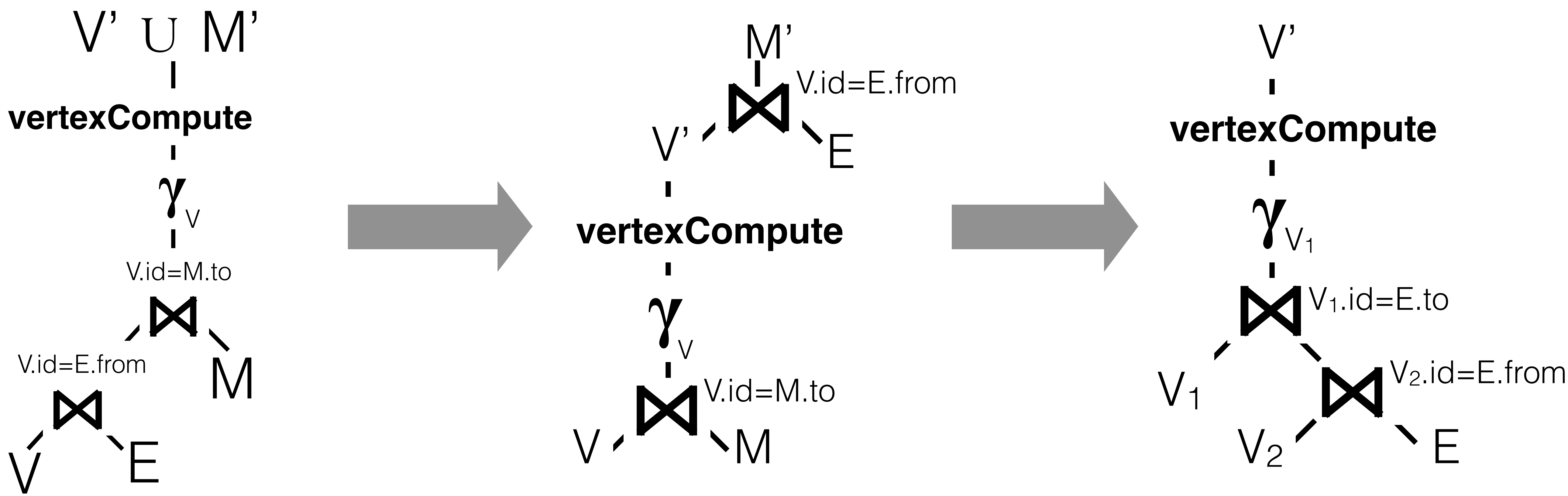


Rewriting Logical Giraph Plan

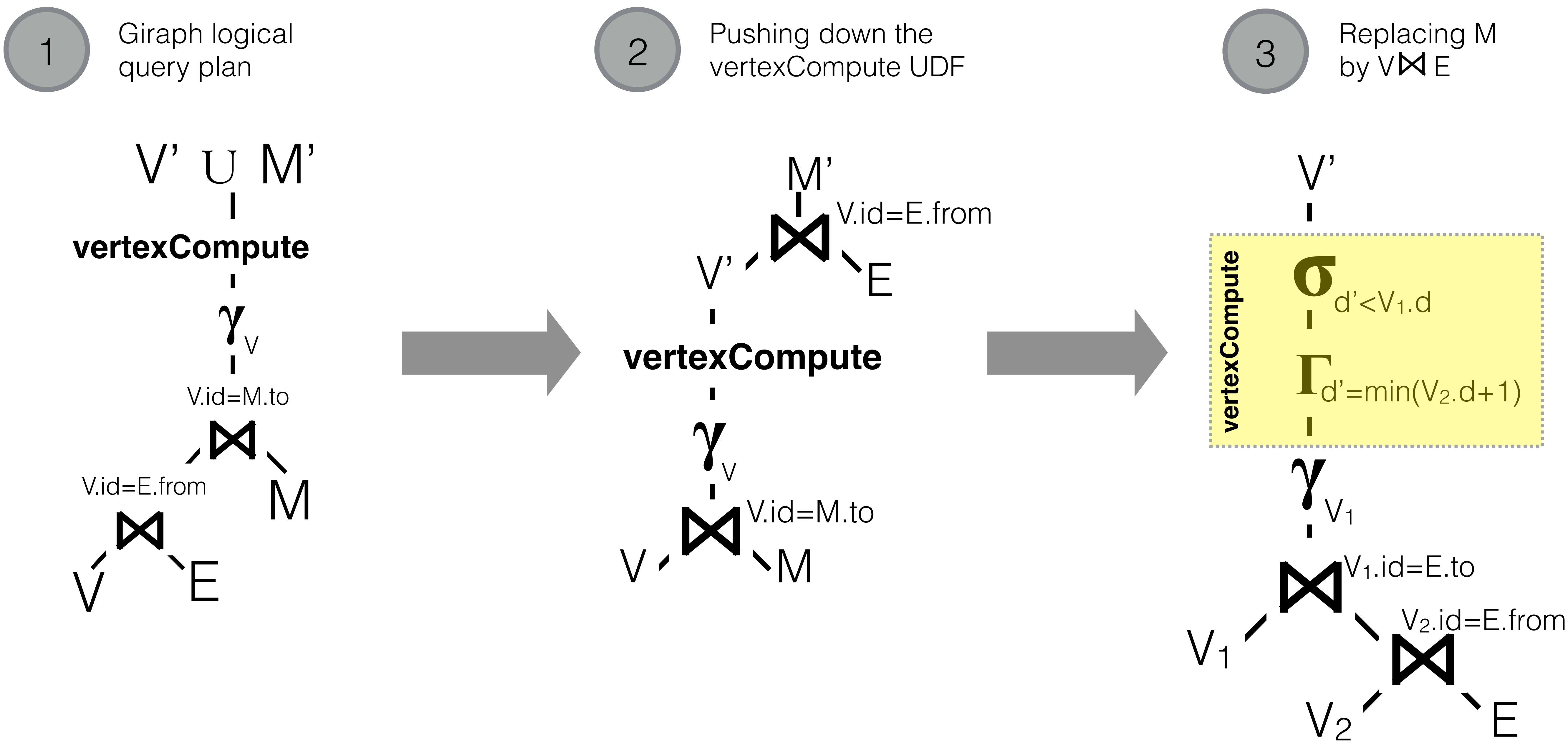
1 Giraph logical query plan

2 Pushing down the vertexCompute UDF

3 Replacing M by V⋈E

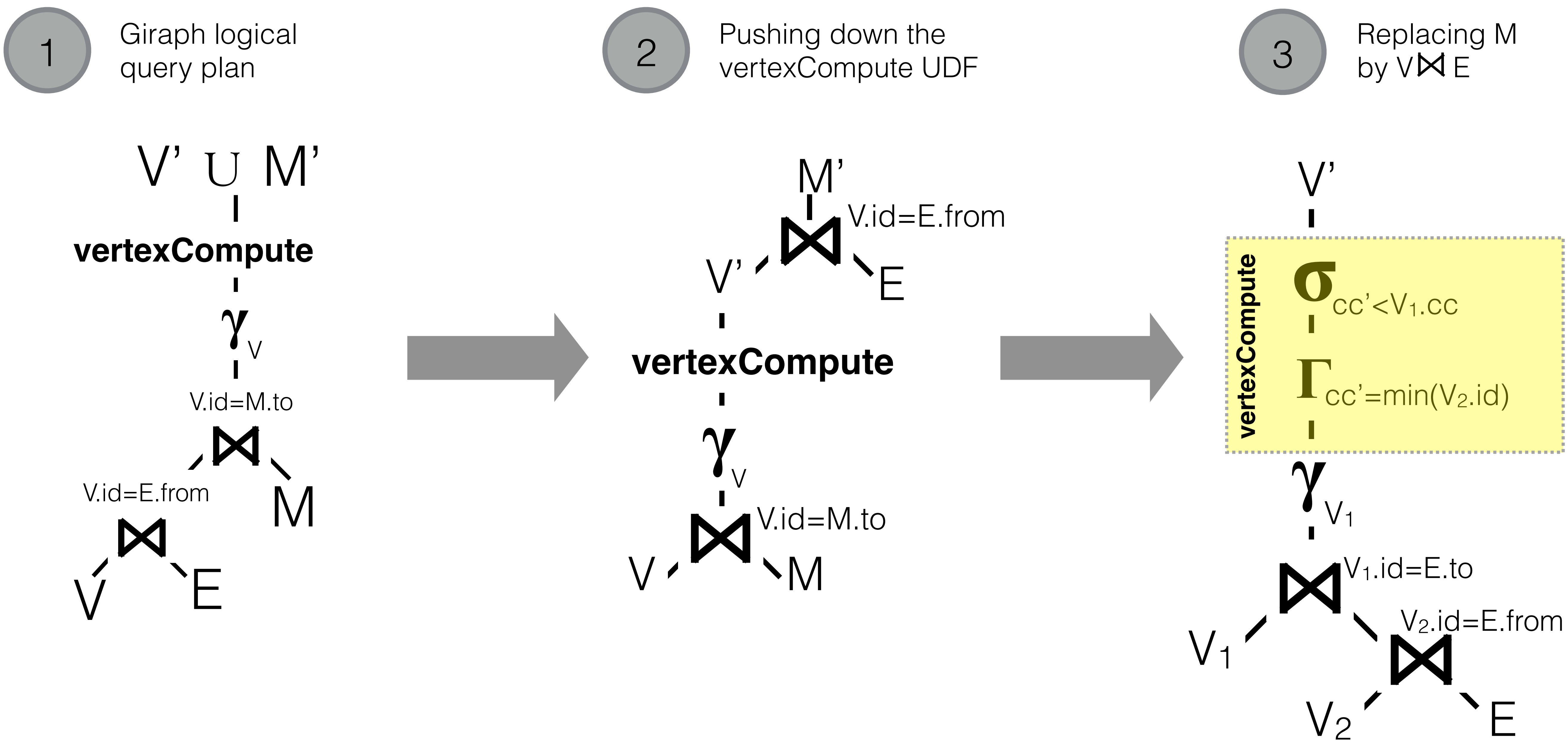


Rewriting Logical Giraph Plan



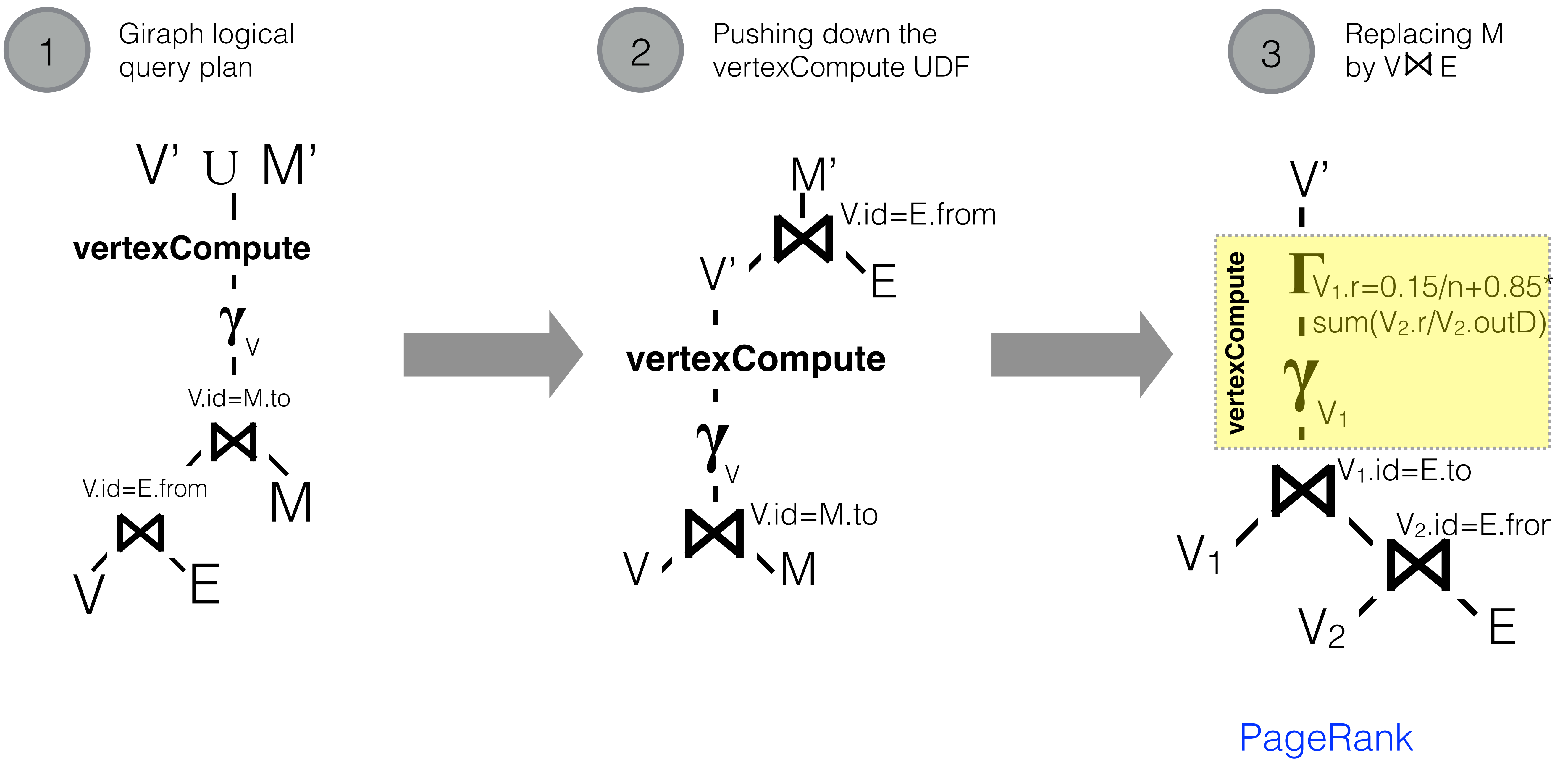
Single Source Shortest Path

Rewriting Logical Giraph Plan

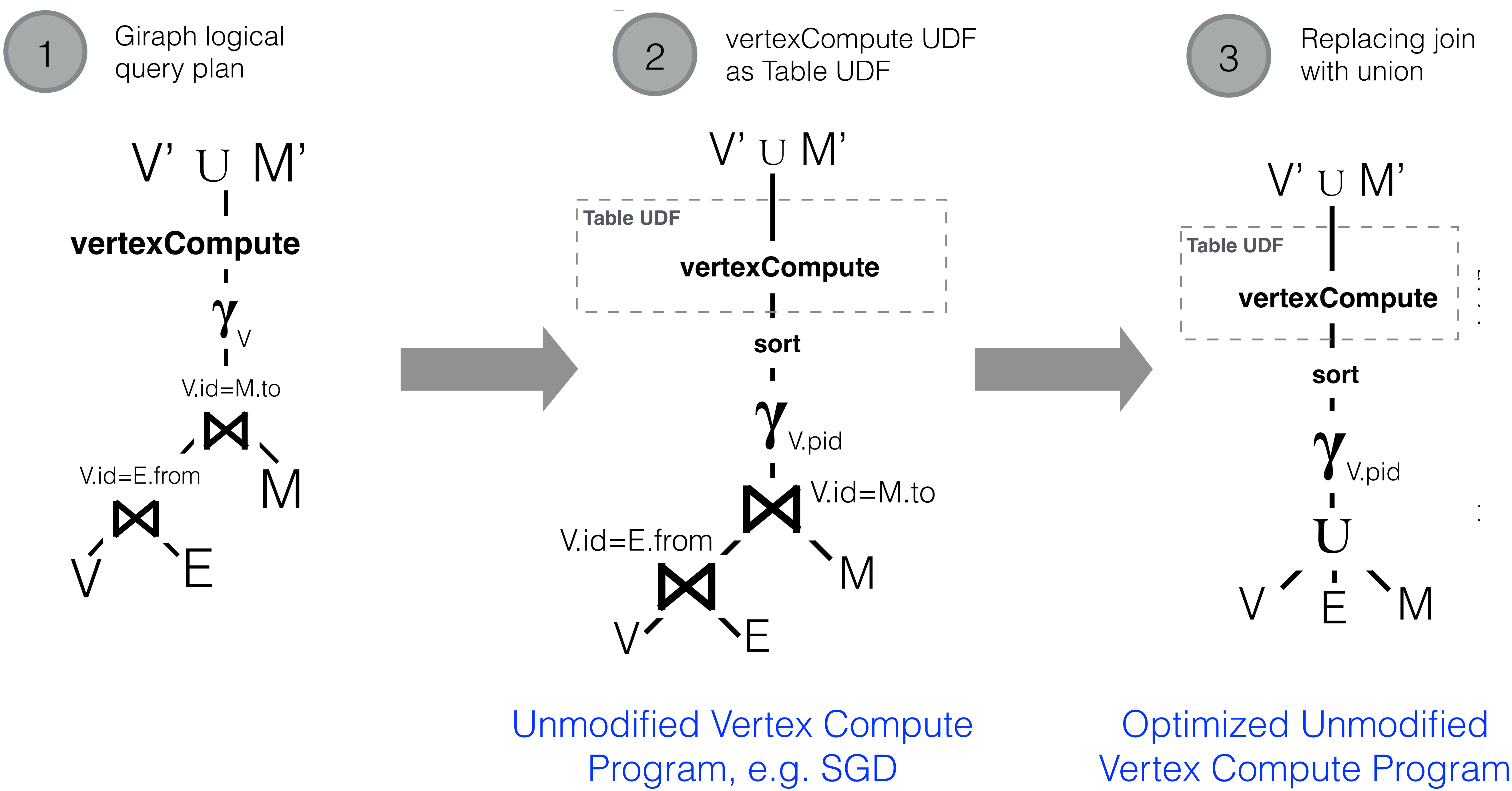


Connected Components

Rewriting Logical Giraph Plan



Rewriting Logical Giraph Plan



2. Graph Query Optimization

Leveraging Relational Query Optimizers

- Multiple rule- or cost-based query rewriting possible; pick the best one using an optimizer
- No hard-coded physical execution plan
- Several new optimizations proposed:
  - update vs replace
  - incremental evaluation
  - join elimination
  - alternate direction graph exploration

Inner Join Update

Updated Input

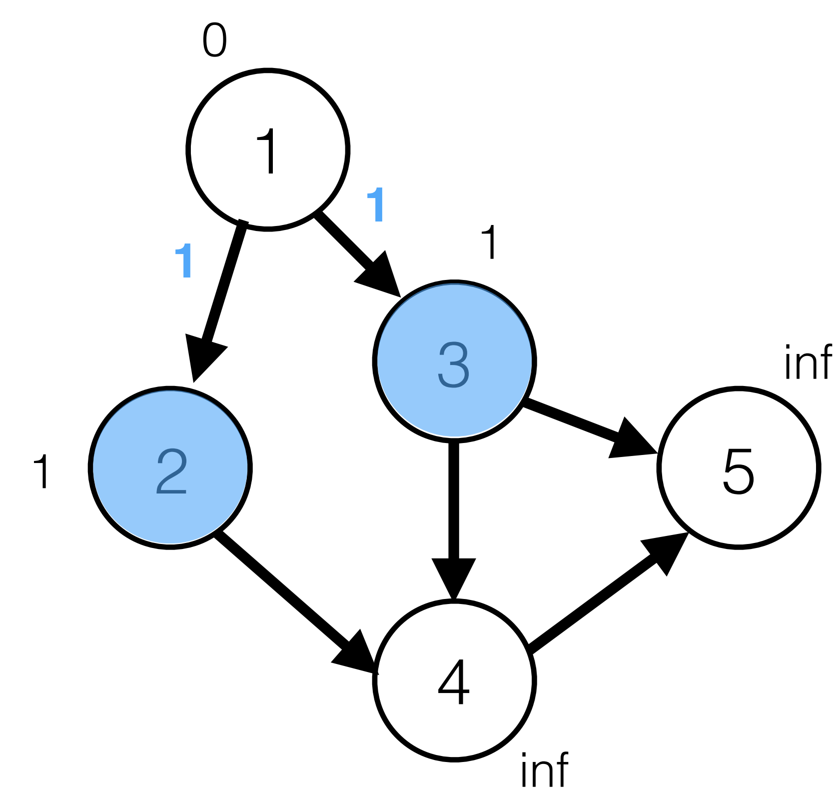
Node	Value
1	0
2	1
3	1
4	inf
5	inf

SSSP

Inner Join

Output

Node	Value
2	1
3	1



Good for small number of updates!

Outer Join Replace

Input

Node	Value
1	0
2	inf
3	inf
4	inf
5	inf

SSSP

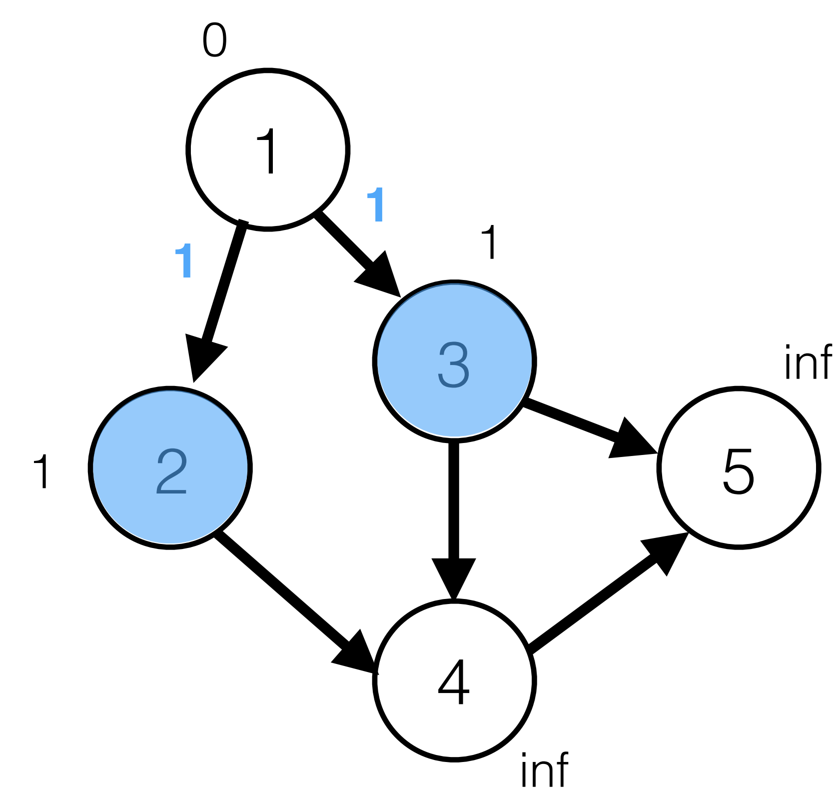
Output

Node	Value
2	1
3	1

Outer Join

New Input

Node	Value
1	0
2	1
3	1
4	inf
5	inf



Good for bulk updates!

Incremental Computation

Inc. Input

Node	Value
1	0

Input

Node	Value
1	0
2	inf
3	inf
4	inf
5	inf

SSSP

Output

Node	Value
2	1
3	1

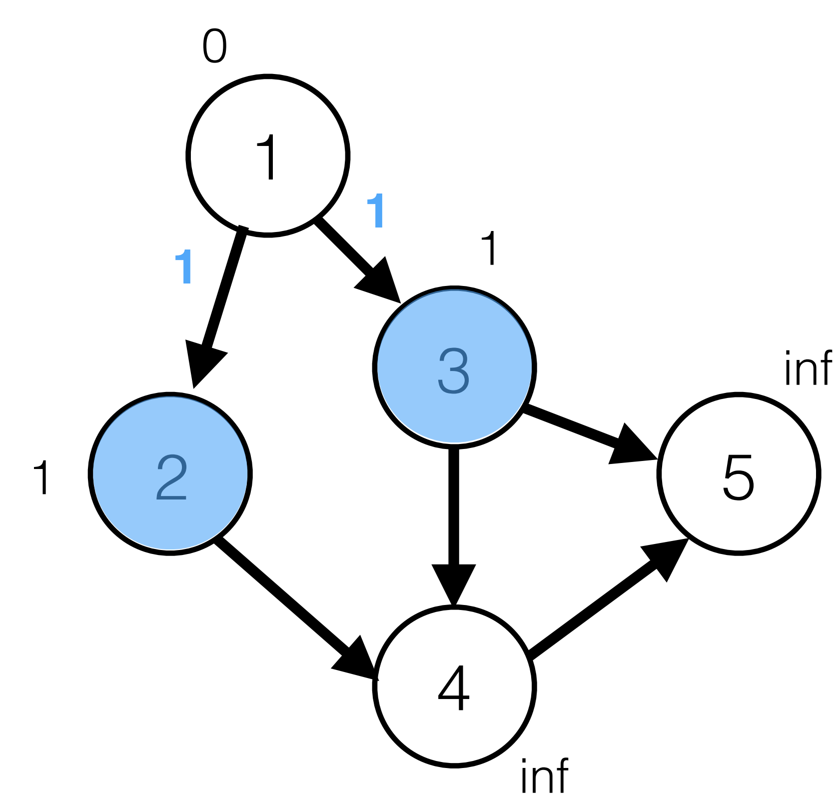
New Inc. Input

Node	Value
2	1
3	1

Outer Join

New Input

Node	Value
1	0
2	1
3	1
4	inf
5	inf



Incremental Computation

Inc. Input

Node	Value
2	1
3	1

SSSP

Output

Node	Value
4	2
5	2

New Inc. Input

Node	Value
4	2
5	2

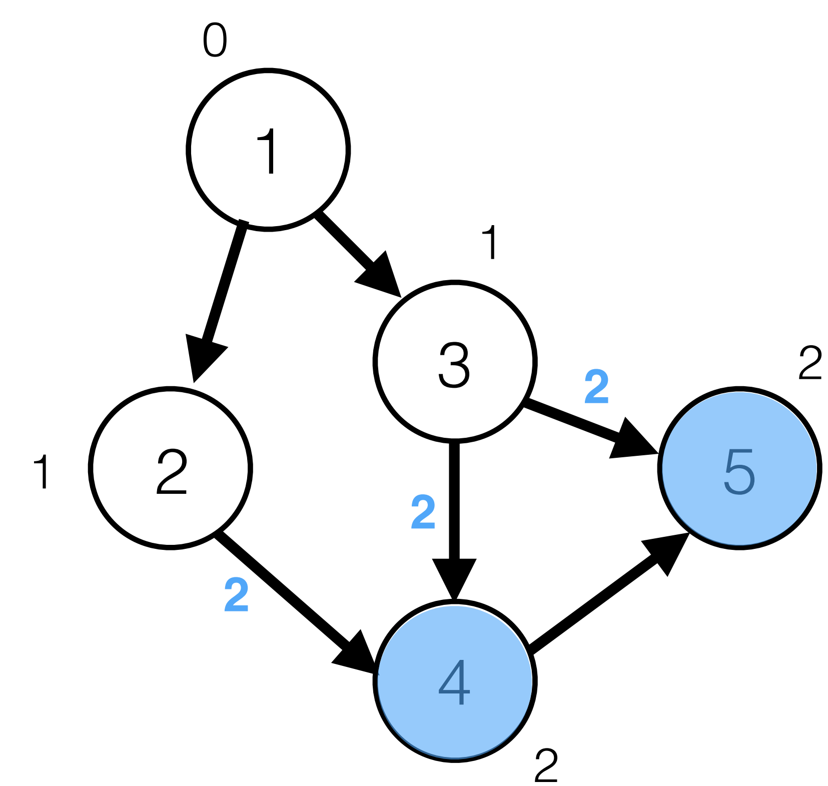
Input

Node	Value
1	0
2	1
3	1
4	inf
5	inf

Outer Join

New Input

Node	Value
1	0
2	1
3	1
4	2
5	2



Faster Iteration Runtime!

3. Column Store Backends

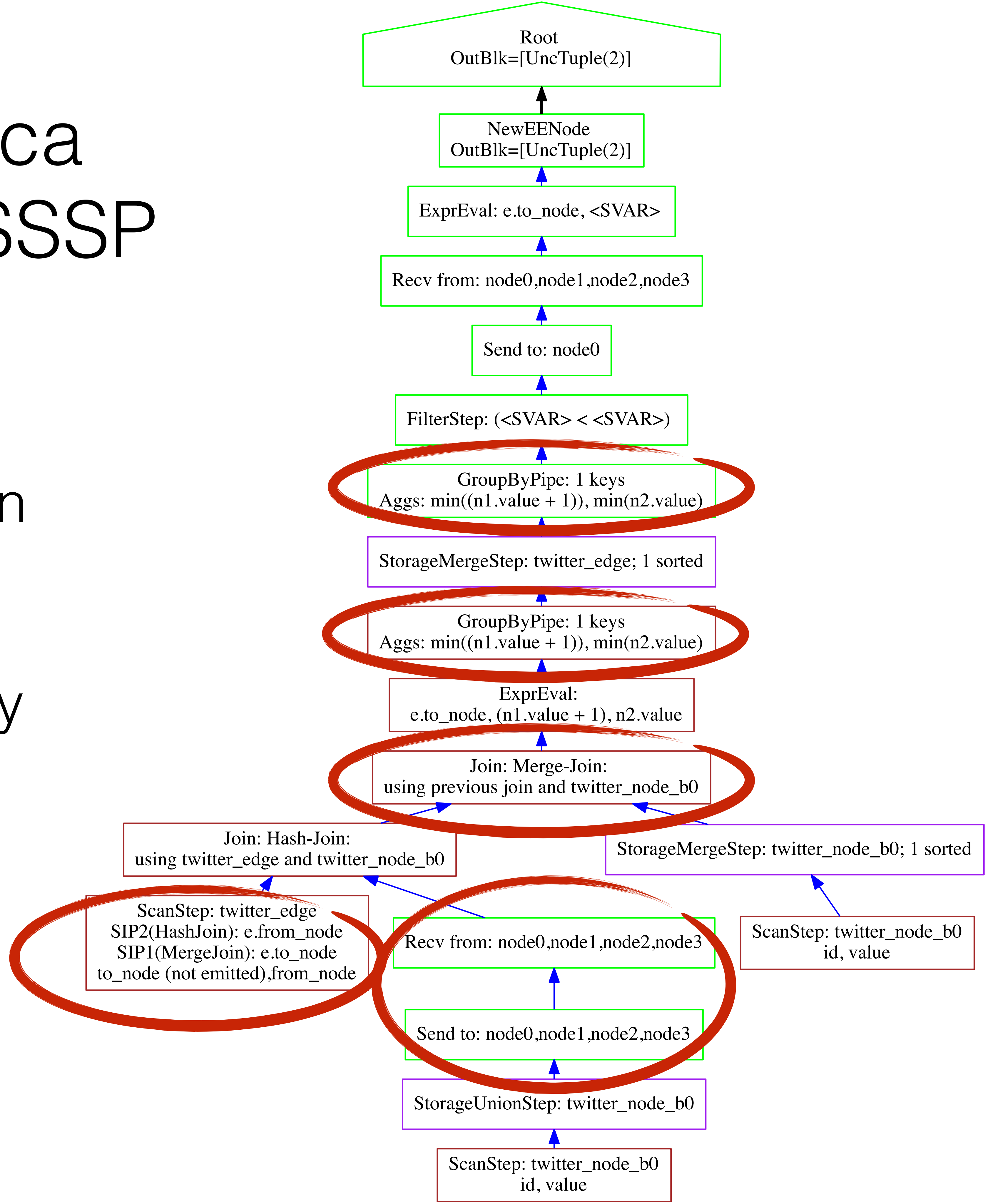
Why columns stores could be a good choice?

- Modern column stores provide several features
  - physical design
  - join optimizations
  - query pipelining
  - intra-query parallelism

- For more details, pick your favorite column store papers:
  - MonetDB
    [Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct, Peter A. Boncz et. al., PVLDB 2009.]
  - C-Store
    [C-Store: A Column-oriented DBMS, Mike Stonebraker et. al., VLDB 2005.]
  - Vertica
    [The Vertica Analytic Database: C-Store 7 Years Later, Andrew Lamb et. al., VLDB 2012.]

Illustration: Vertica Query Plan for SSSP

- Early filtering using sideways information passing
- Fully pipelined query execution
- Picks the right join strategies, e.g. broadcast



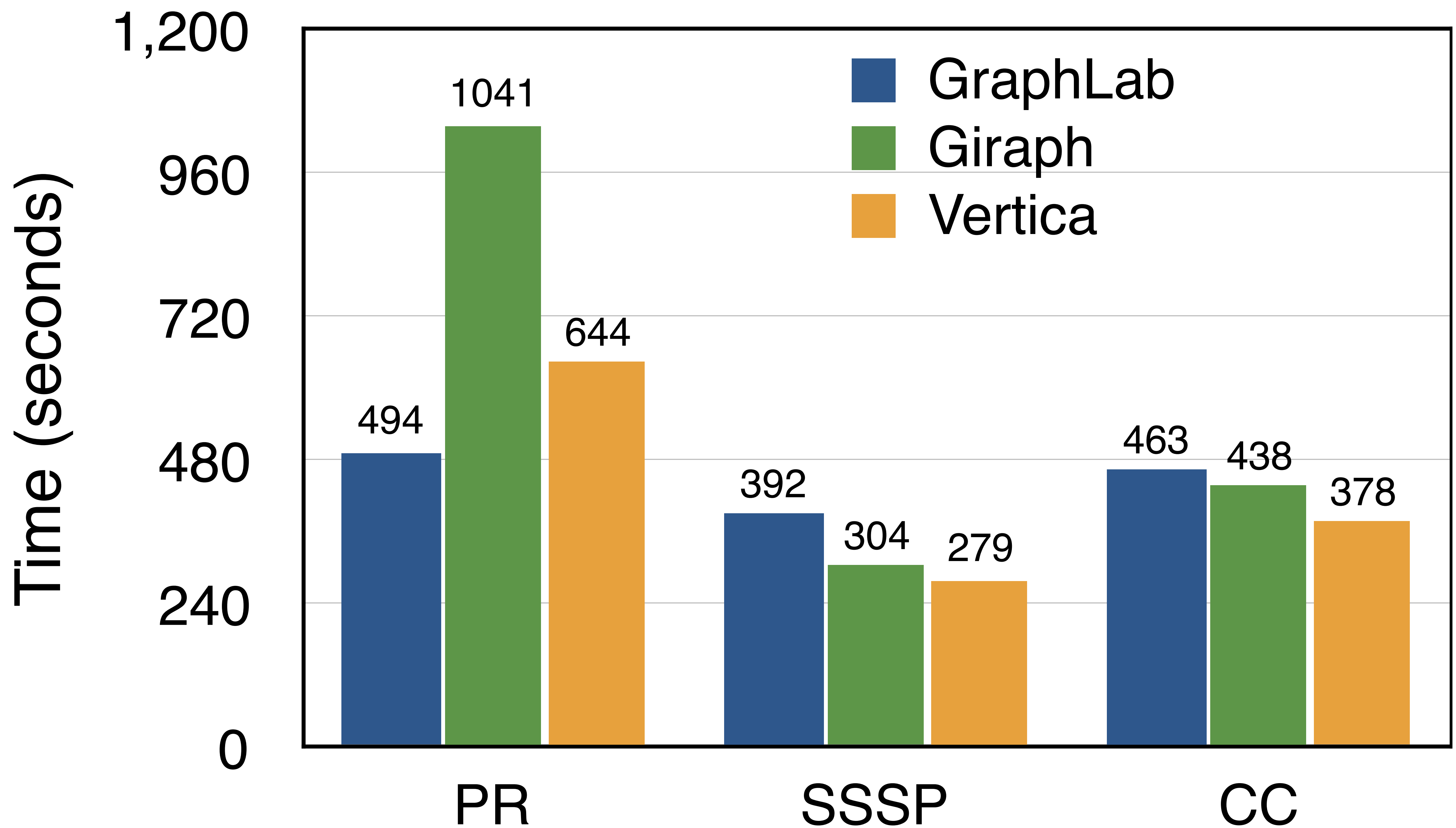
4. Comparison with Specialized Graph Systems

Setup

- Systems:
  - Vertica
  - Giraph
  - GraphLab
- Datasets:
  - directed (Twitter, LiveJournal)
  - undirected (Youtube, LiveJournal)
- Machines
  - 4 machines (12 cores, 48GB memory, 1.4TB disk)
- Data preparation
  - upload time [Vertica: 916s; GraphLab: 472s; Giraph: 268s]
  - disk usage [Vertica: 10GB; GraphLab/Giraph: 73GB]

Typical Graph Analytics

Twitter graph: 1.4 billion edges, 41.6 million nodes



Advanced Graph Analytics

- Mixing graph and relational queries

Twitter graph with synthetic metadata

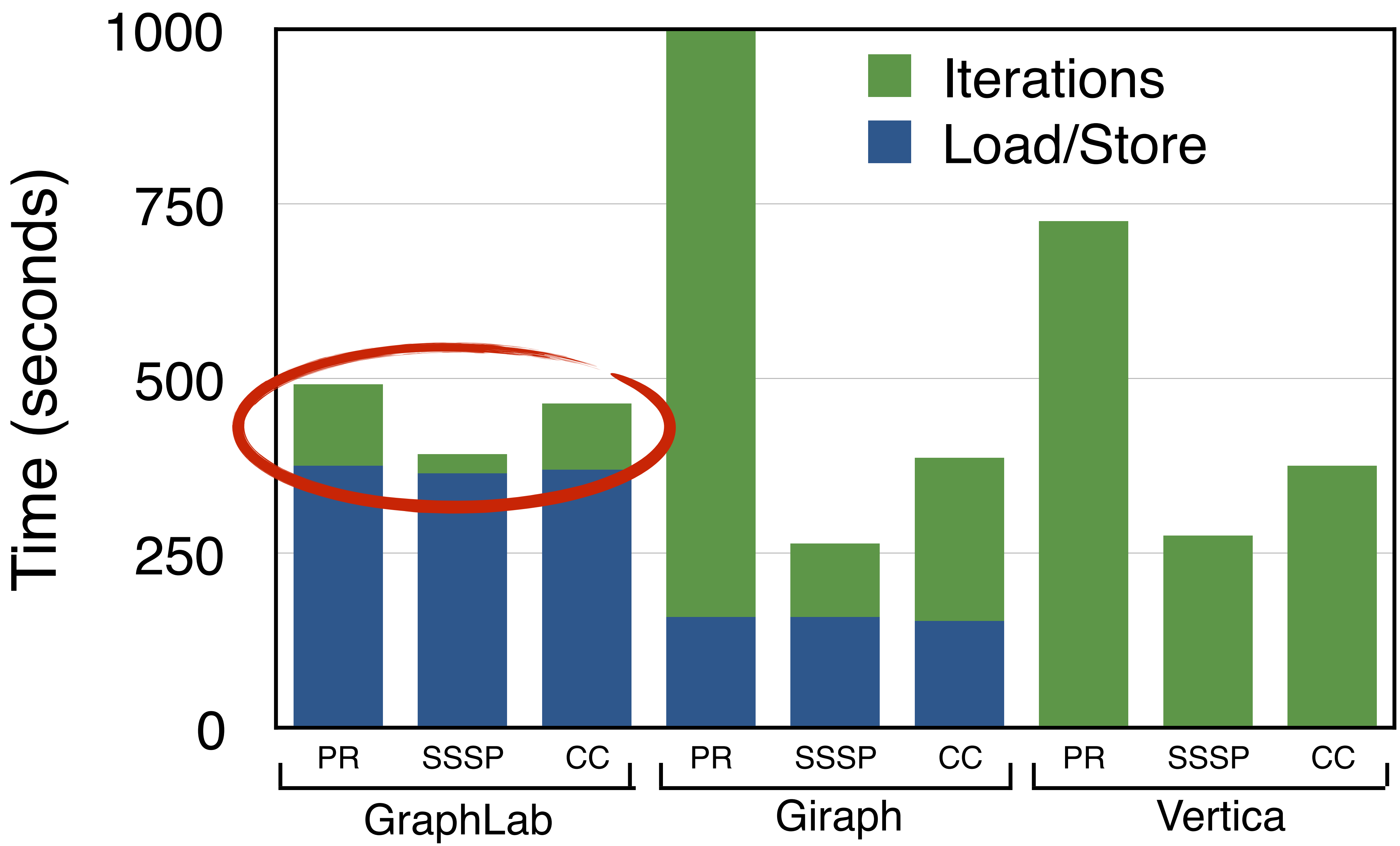
Query	**Type**	**Vertica**	**Giraph**	**SpeedUp**
Sub-graph Projection & Selection	PR	55.6	954.6	17.2
	SSSP	101.3	405.5	4.0
Graph Analysis Aggregation	PR	643.9	1089.7	1.7
	SSSP	279.8	349.9	1.3
Graph Joins	PR+SSSP	927.0	1435.9	1.5

- Multi-hop neighborhood queries

Query	**Dataset**	**Vertica**	**Giraph**
Strong Overlap	Youtube	259.56	230.01
	LiveJournal-undir	381.05	out of memory
Weak Ties	Youtube	746.14	out of memory
	LiveJournal-undir	1,475.99	out of memory

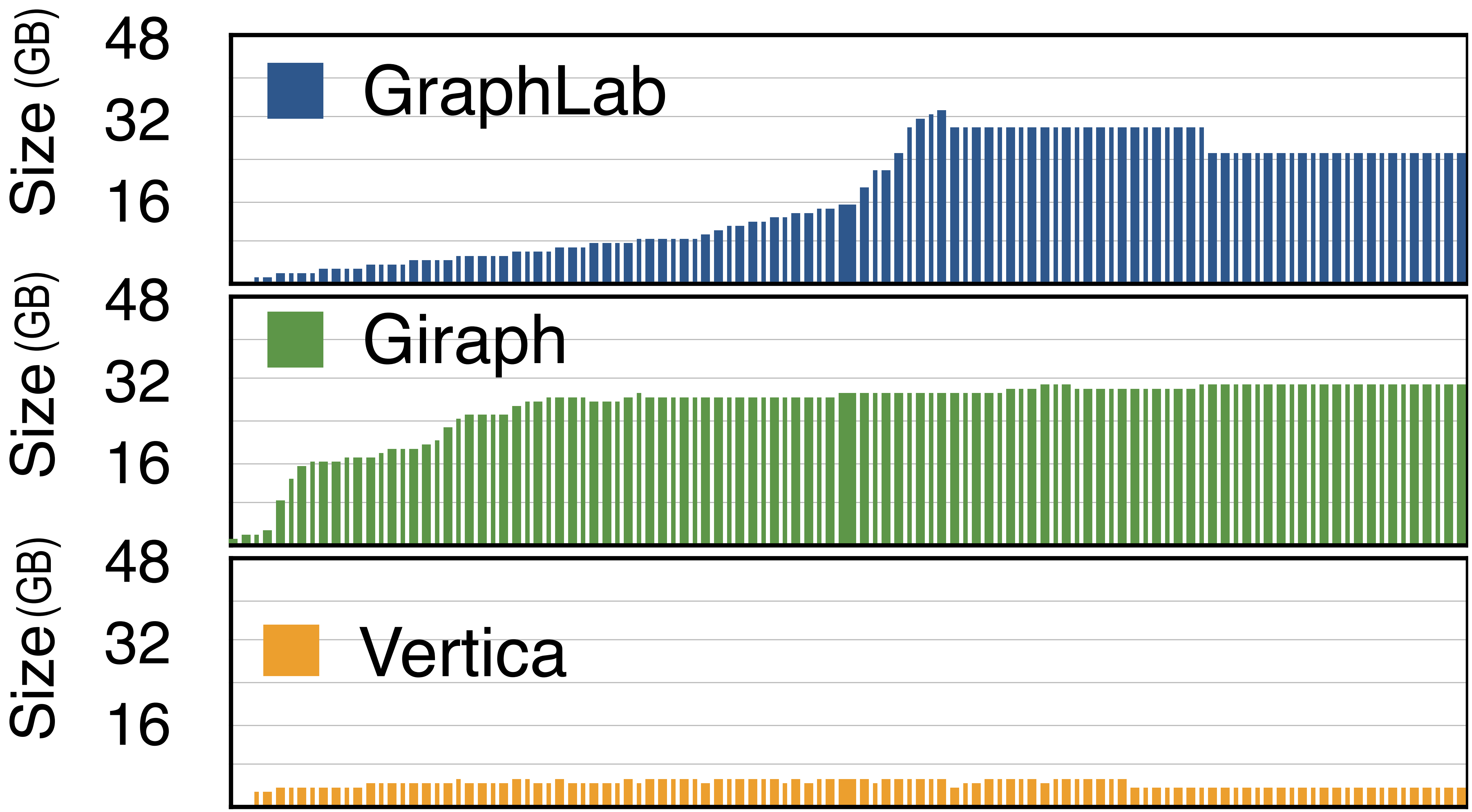
Detailed Analysis: Cost Breakdown

Twitter graph: 1.4 billion edges, 41.6 million nodes



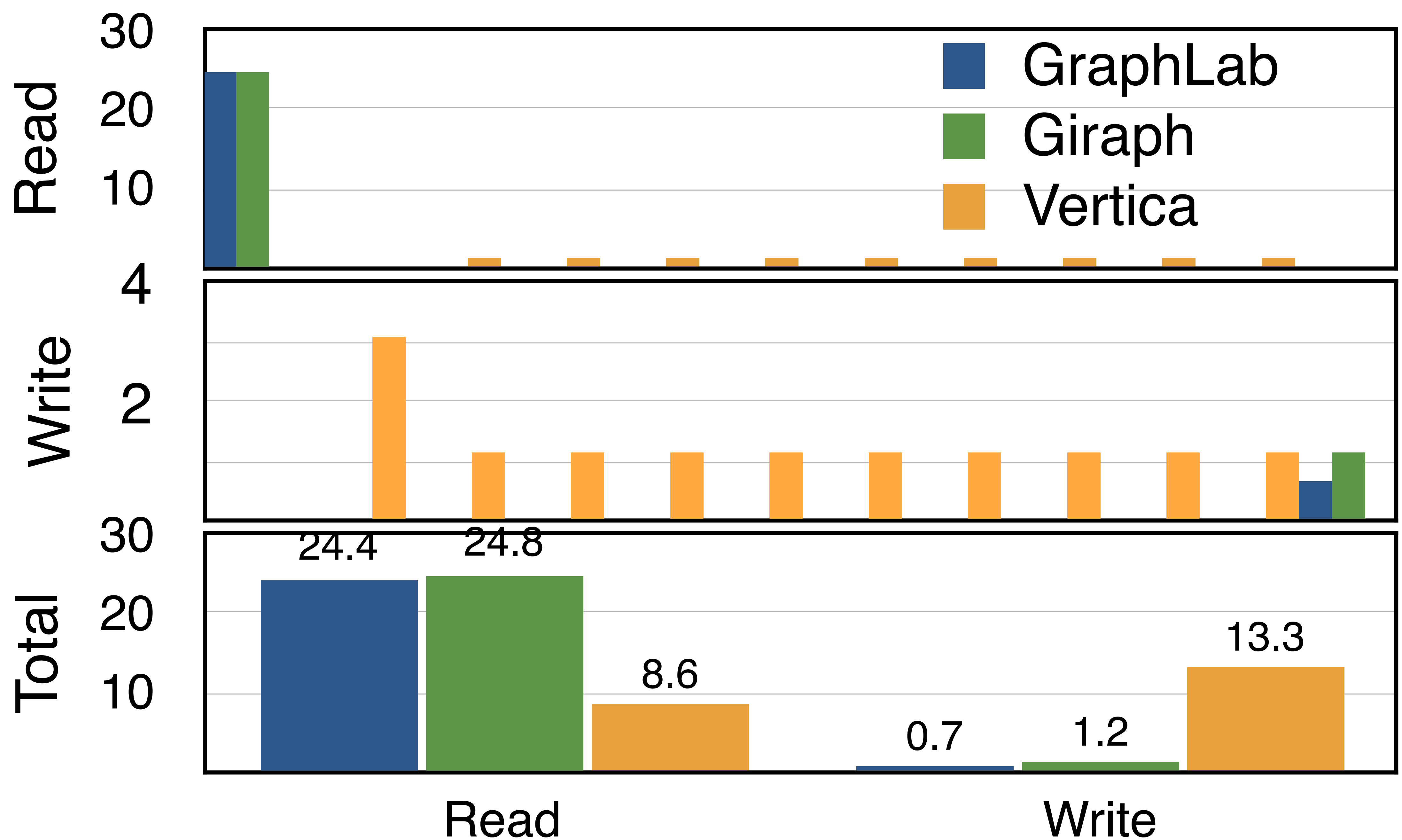
Detailed Analysis: Memory Footprint

Twitter graph: 1.4 billion edges, 41.6 million nodes



Detailed Analysis: I/O Footprint

Twitter graph: 1.4 billion edges, 41.6 million nodes



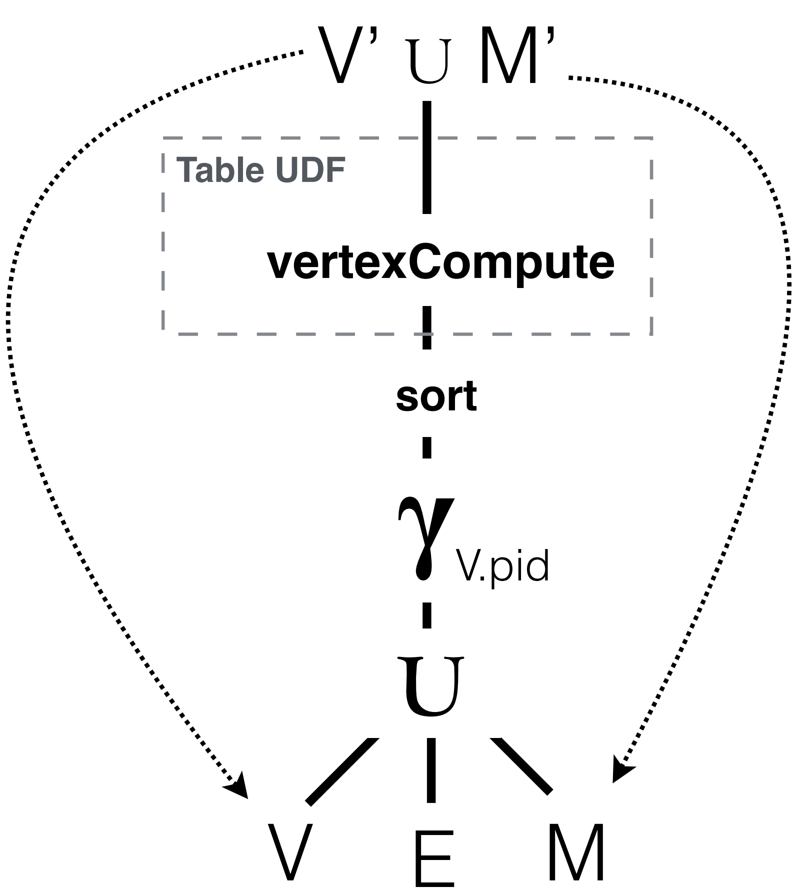
Problem: significantly high I/O

Can we do better?

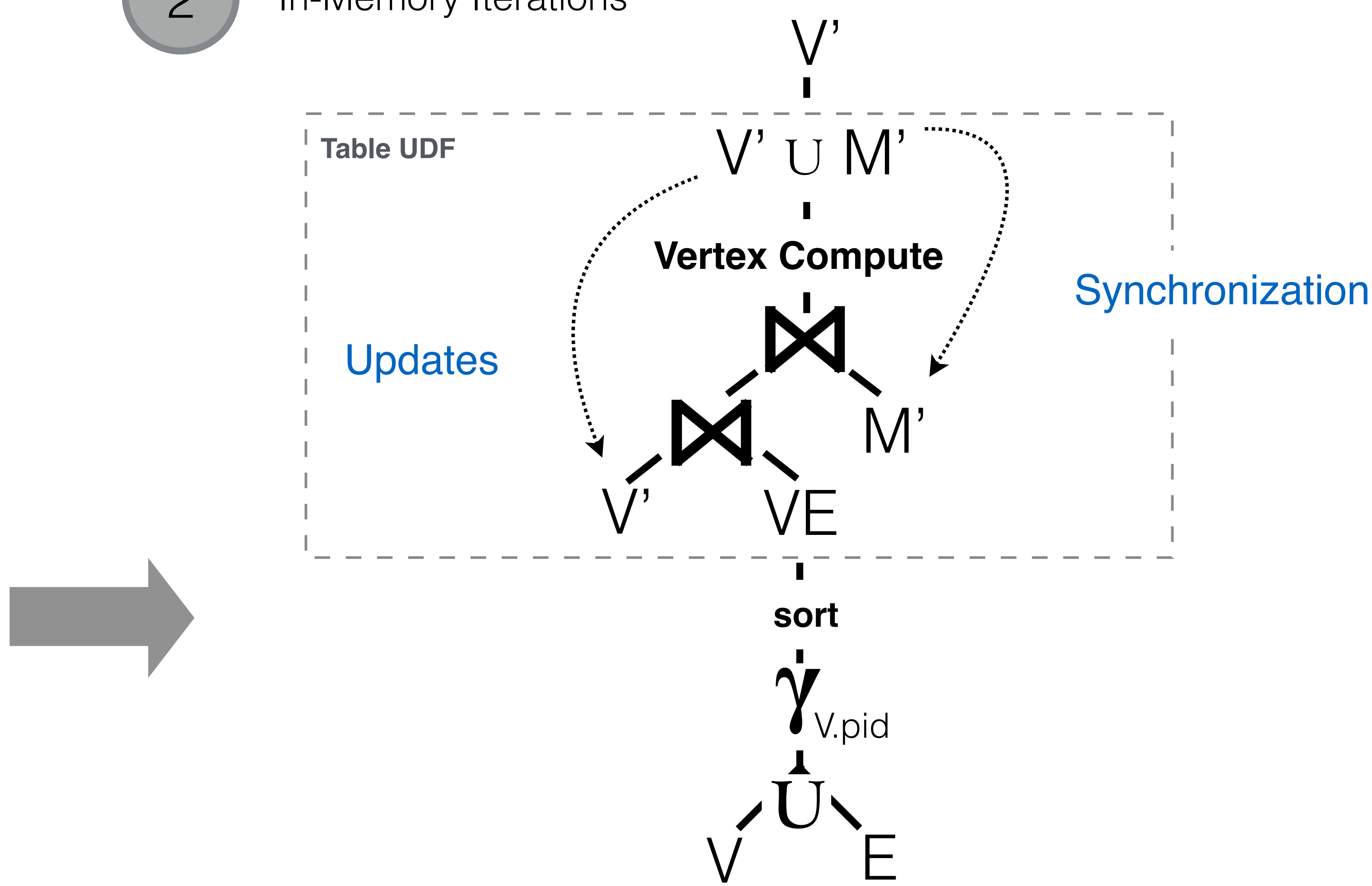
5. Extending Column Stores

Rewriting Graph Query Plan (Yet again!)

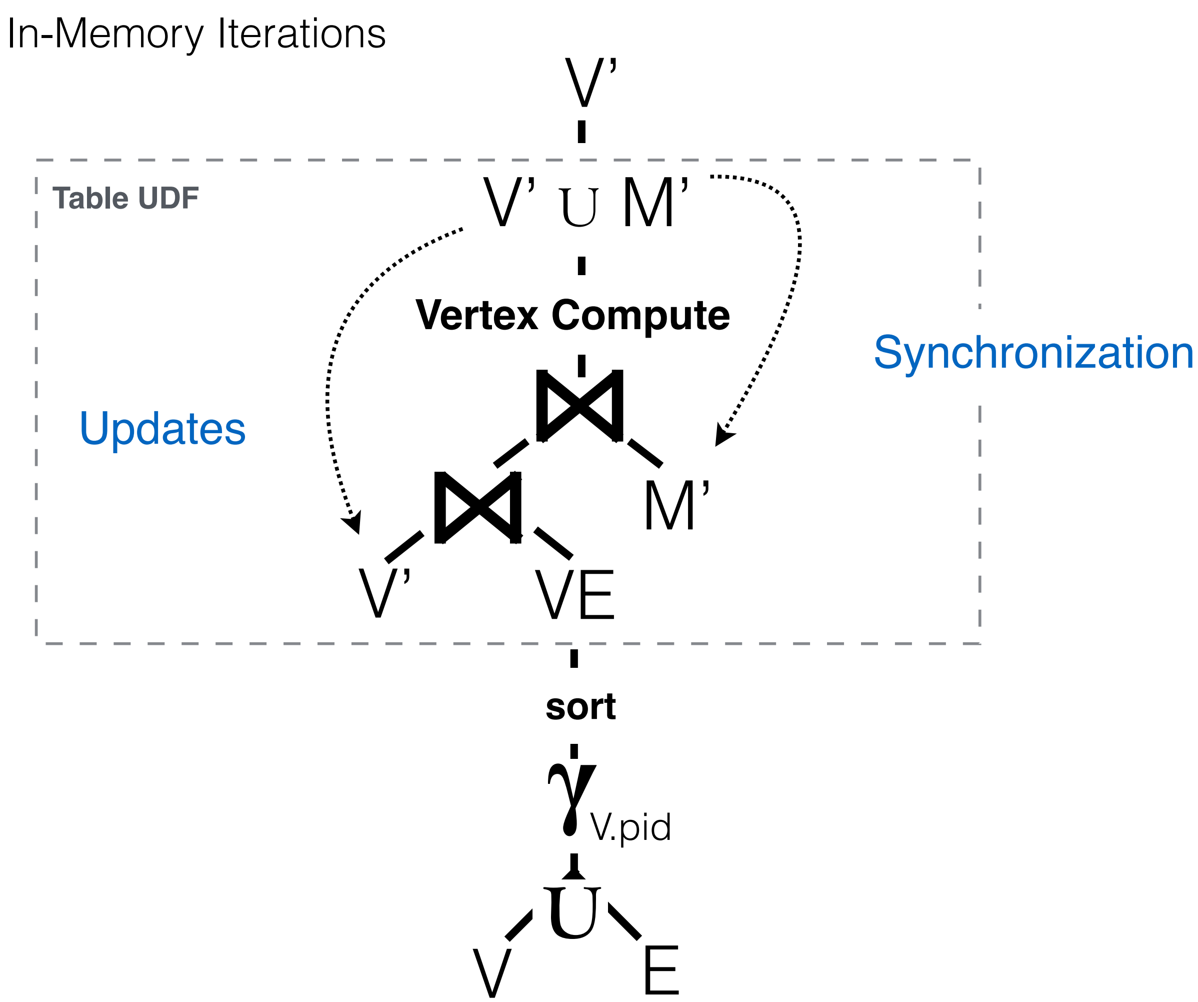
1 Disk-based Iterations



2 In-Memory Iterations



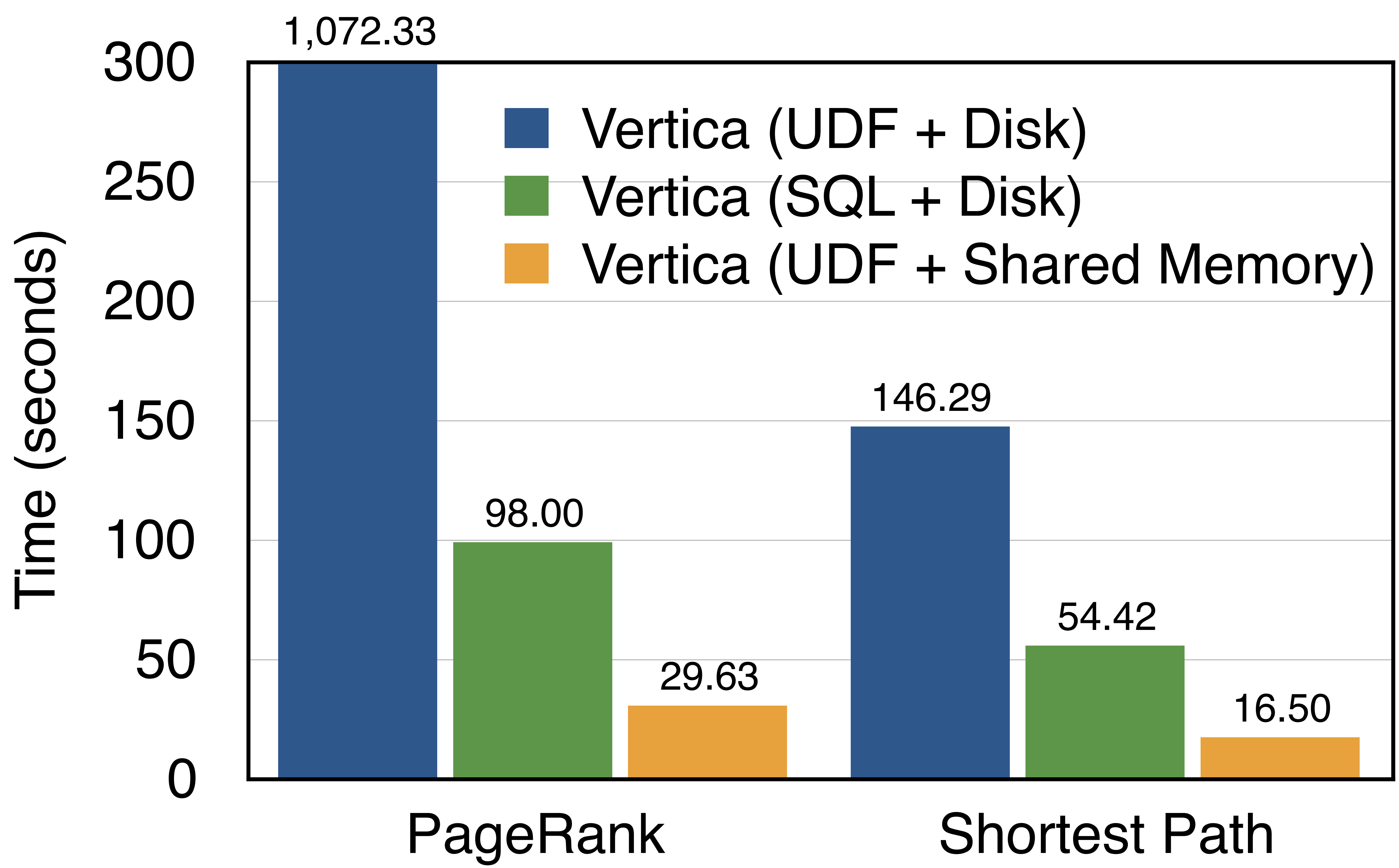
Trading Memory for I/O



- Loading and keeping data in main-memory — no additional I/Os for each iteration
- All iterations run as a single transaction — reduce overheads such as logging, locking, buffer lookups
- Unmodified vertex-program run via table UDFs
- Communication (message passing) via shared memory

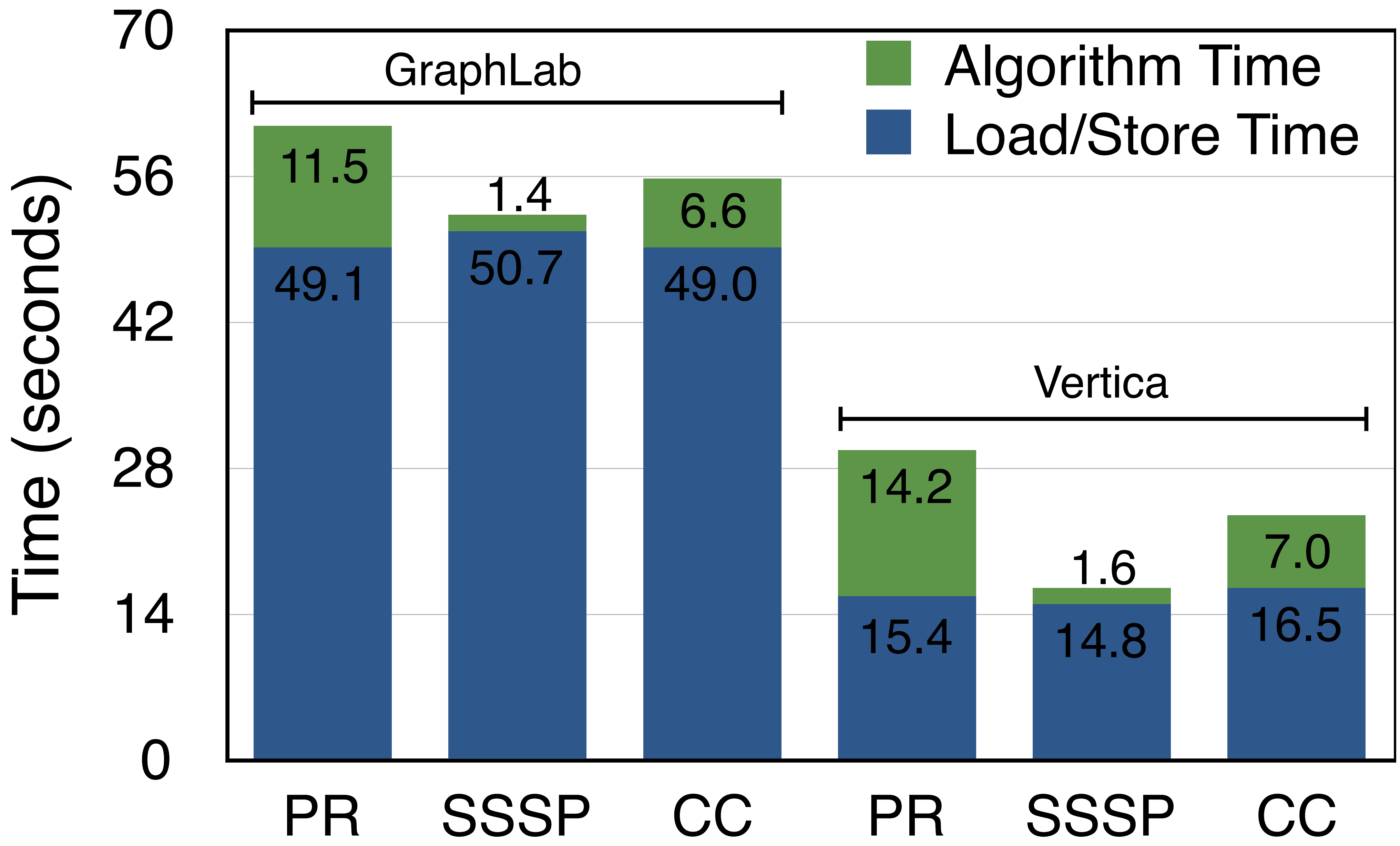
Comparing Different Implementations in Vertica

LiveJournal graph: 69 million edges, 4.8 million nodes



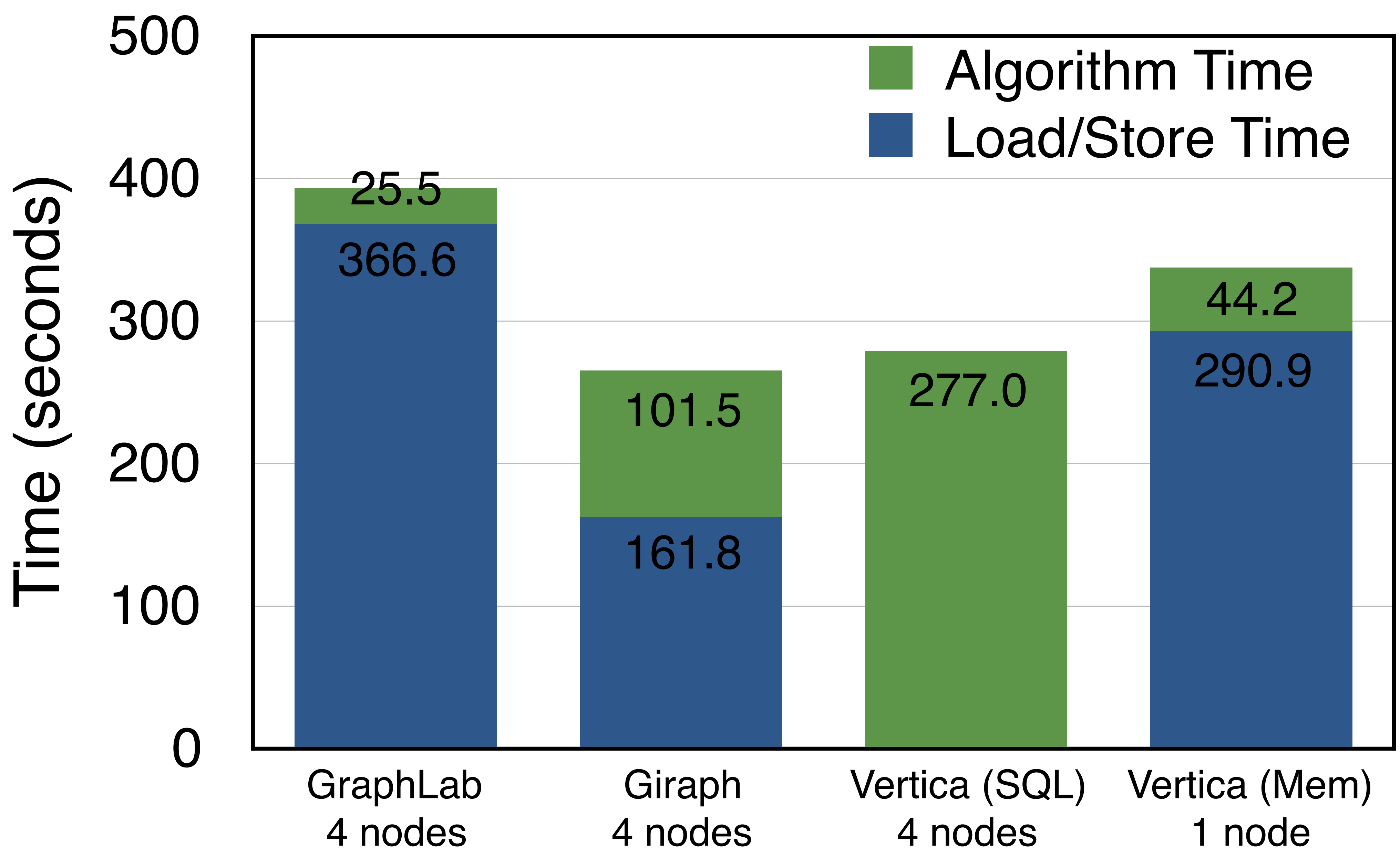
Comparison with GraphLab

LiveJournal graph: 69 million edges, 4.8 million nodes



Scaling to larger graphs

Twitter graph: 1.4 billion edges, 41.6 million nodes



Conclusion

- Efficient graph analytics possible within column stores such as Vertica
  - graph queries can be mapped to SQL
  - several query optimizations can be applied
  - column stores serve as efficient backends
  - could extend column stores to trade memory for I/O

- The curious case of relational database re-discovery
  - repeatedly emerged as the backend for several new data/applications, e.g., XML, RDF, Spatial, Array, etc.
  - cycles of branch-innovate-merge-commit

- Next time you have a big data problem —> try relational databases!