NEDB Day 2012

G*

A Parallel System for Efficiently Managing Large Graphs

Jeong-Hyon Hwang, Sean Spillane, Daniel Bokser,

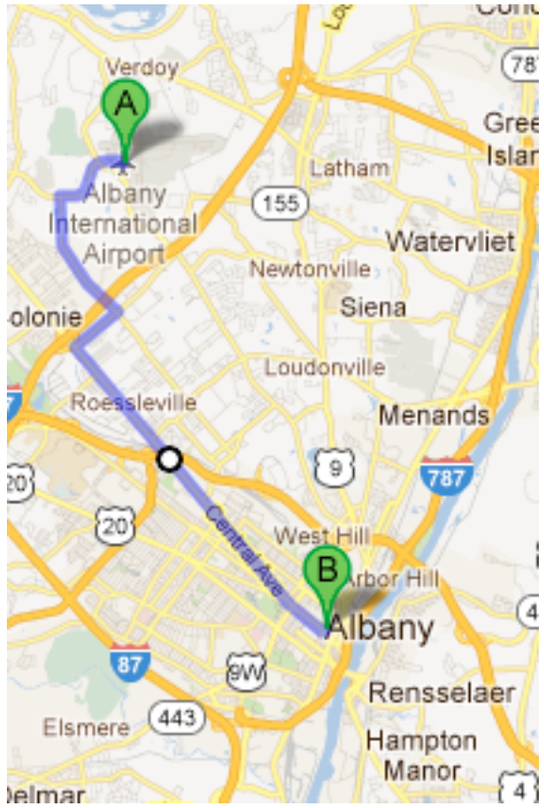
Daniel Kemp, Jayadevan Vijayan, Jeremy Birnbaum

State University of New York - Albany

Graph Data Analytics

- Transportation

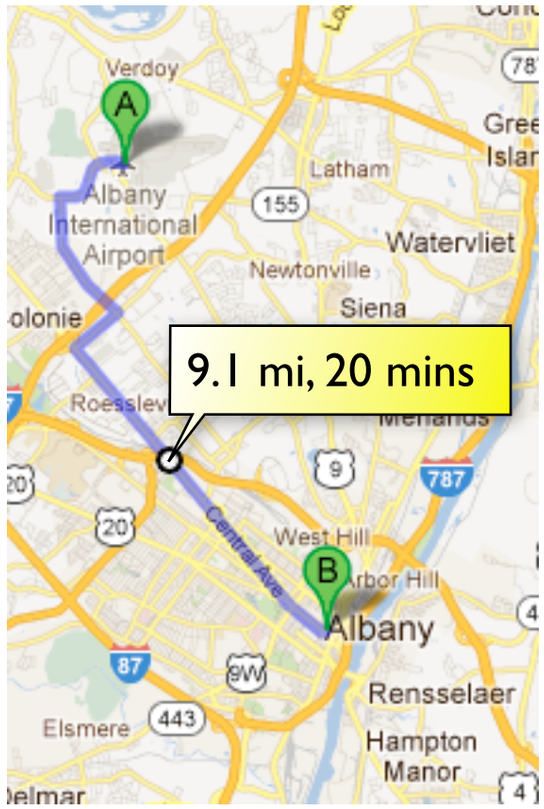
5:00 AM - 6:00 AM



Graph Data Analytics

- Transportation

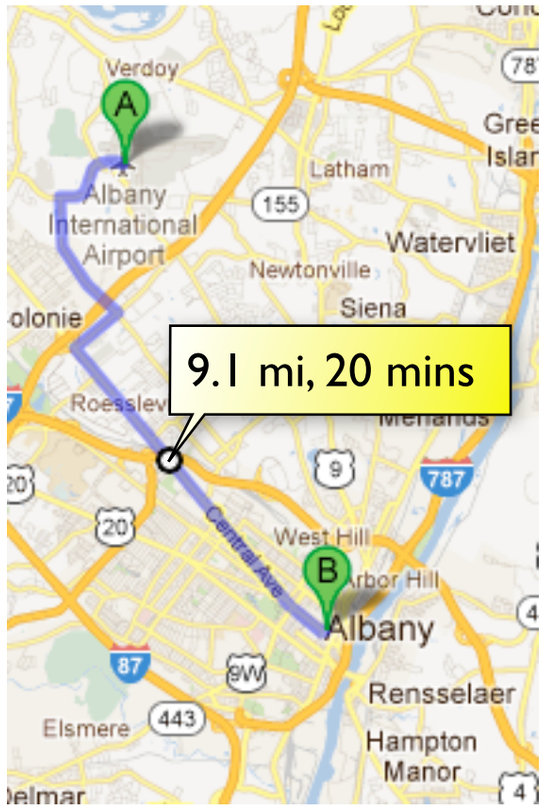
5:00 AM - 6:00 AM

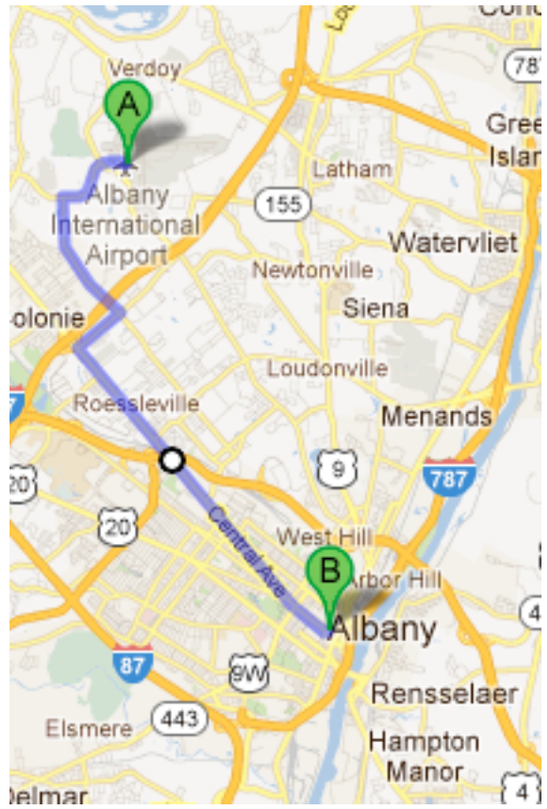


Graph Data Analytics

- Transportation

5:00 AM - 6:00 AM

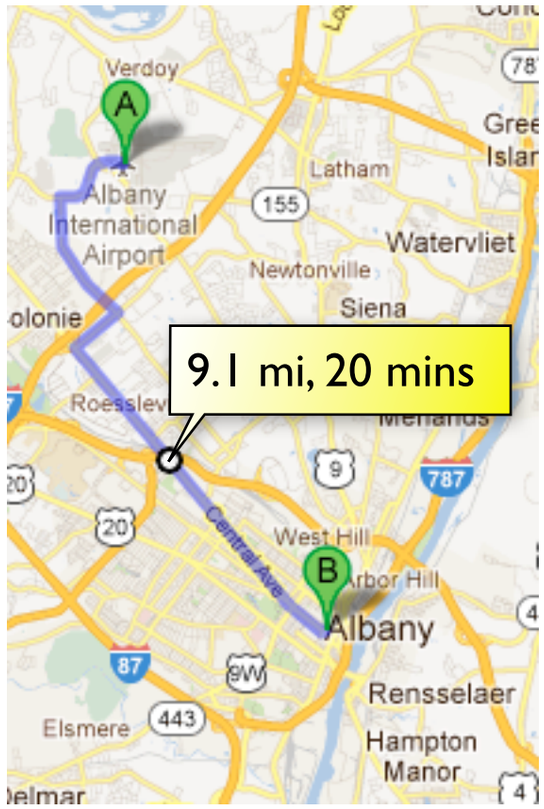




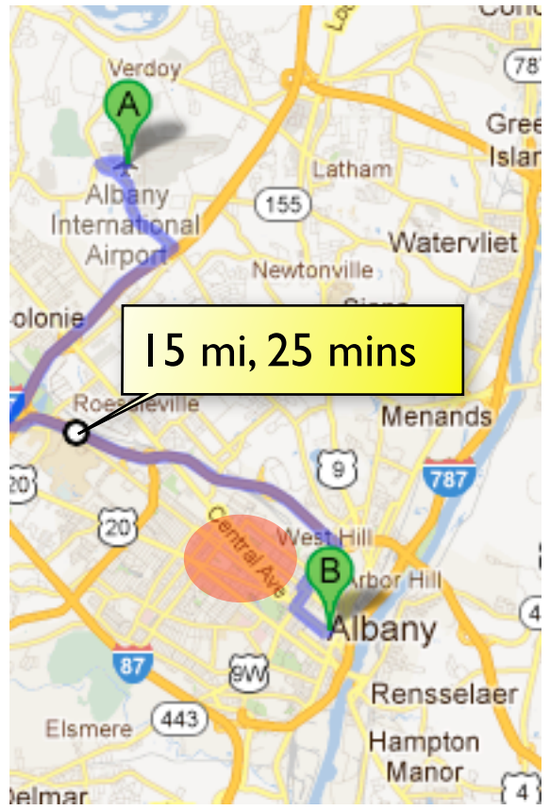
Graph Data Analytics

- Transportation

5:00 AM - 6:00 AM



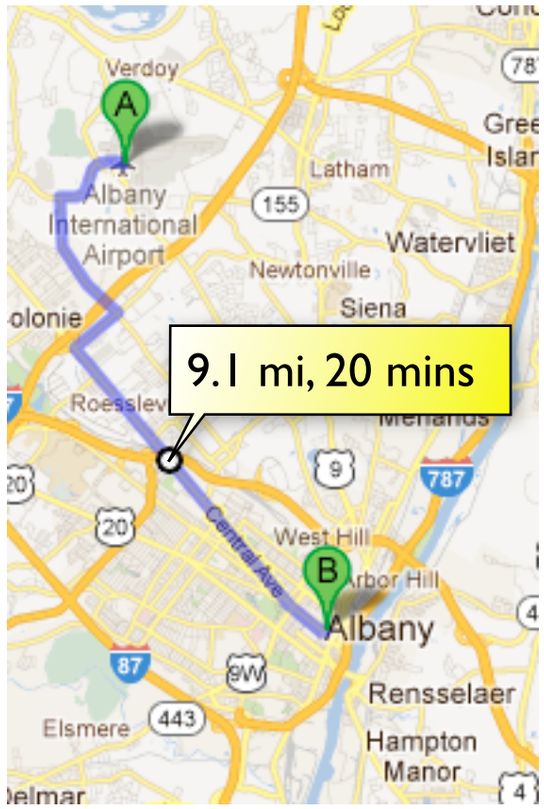
6:00 AM - 7:00 AM



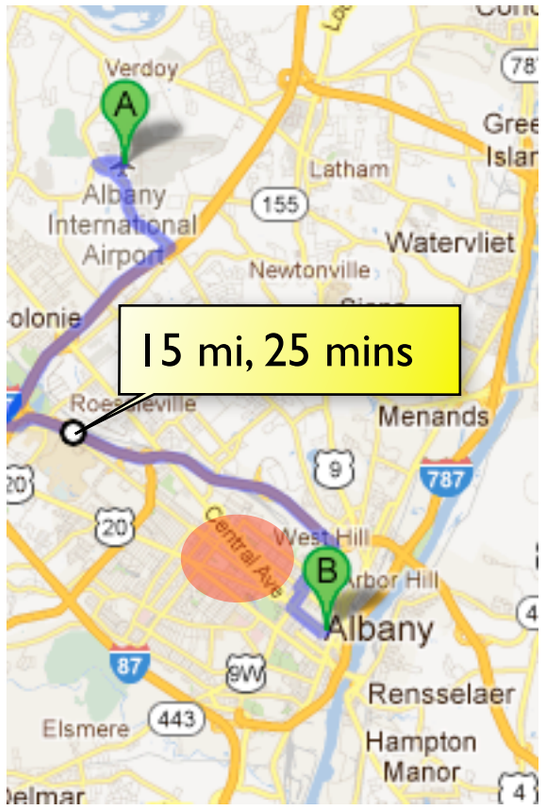
Graph Data Analytics

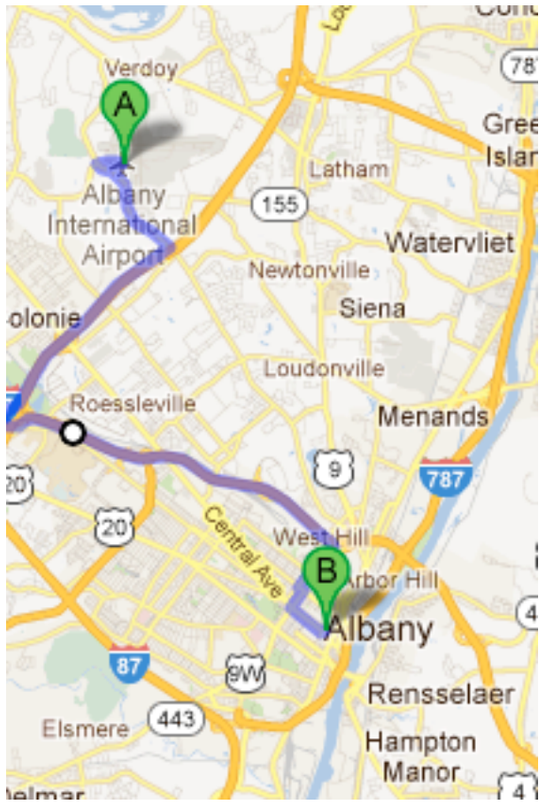
- Transportation

5:00 AM - 6:00 AM



6:00 AM - 7:00 AM

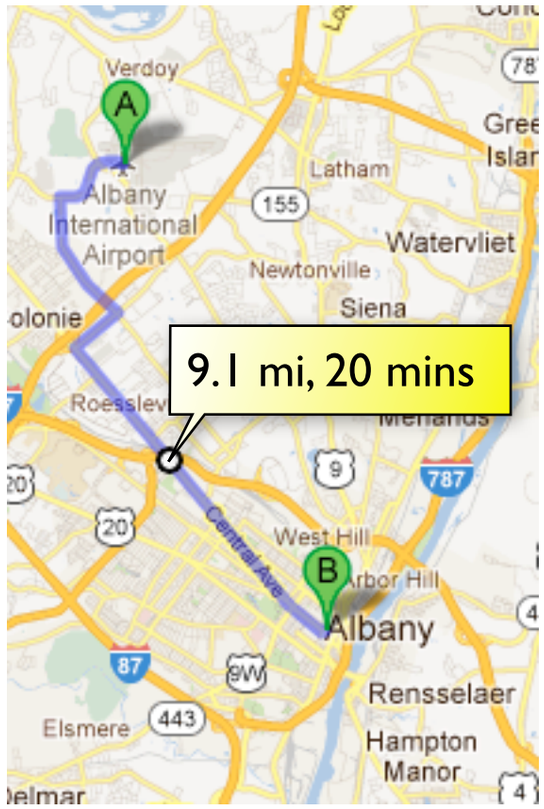




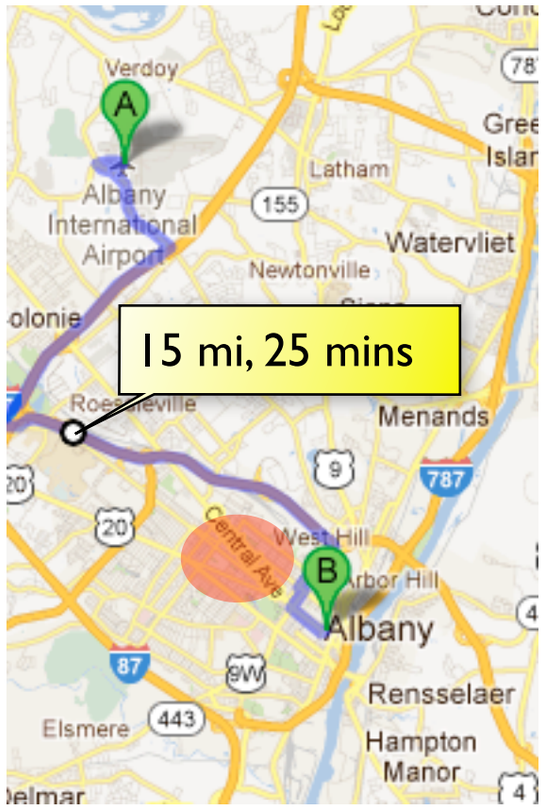
Graph Data Analytics

- Transportation

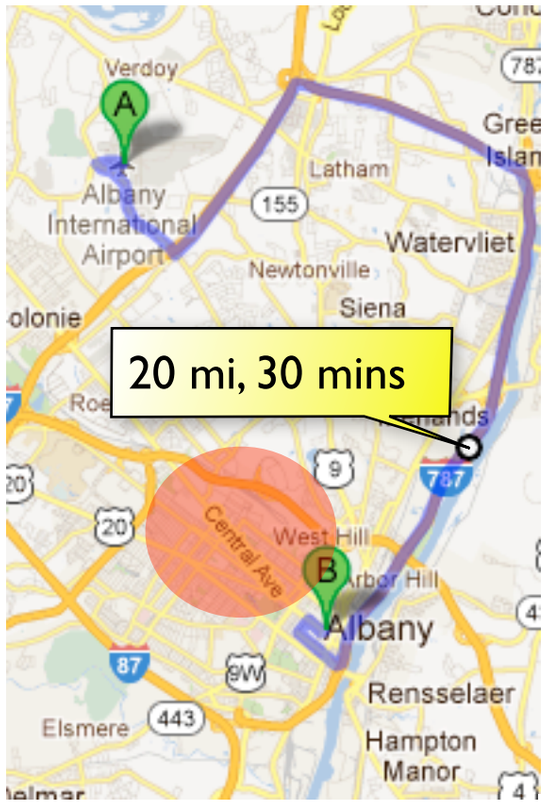
5:00 AM - 6:00 AM



6:00 AM - 7:00 AM



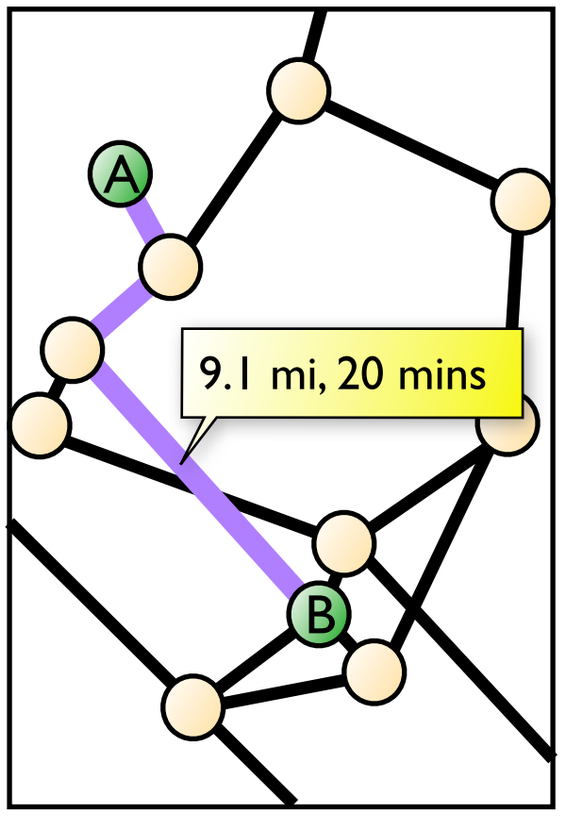
7:00 AM - 8:00 AM



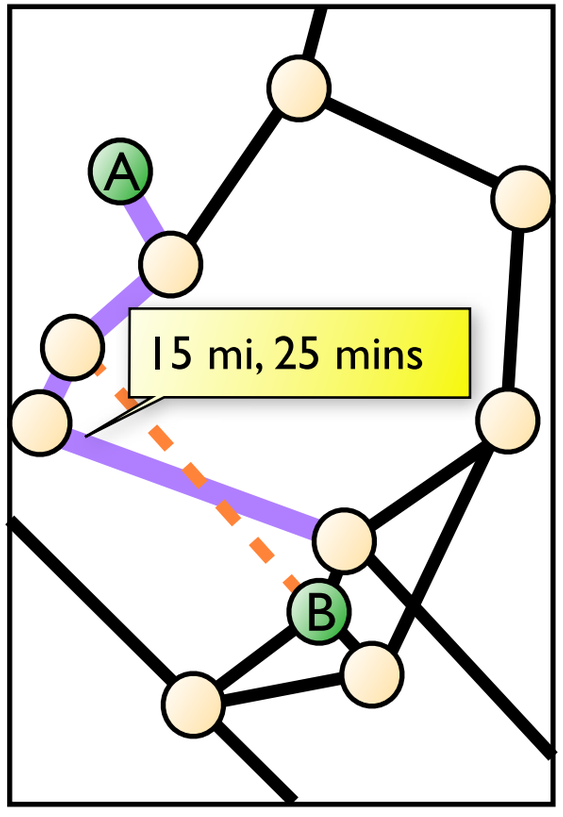
Graph Data Analytics

- Transportation

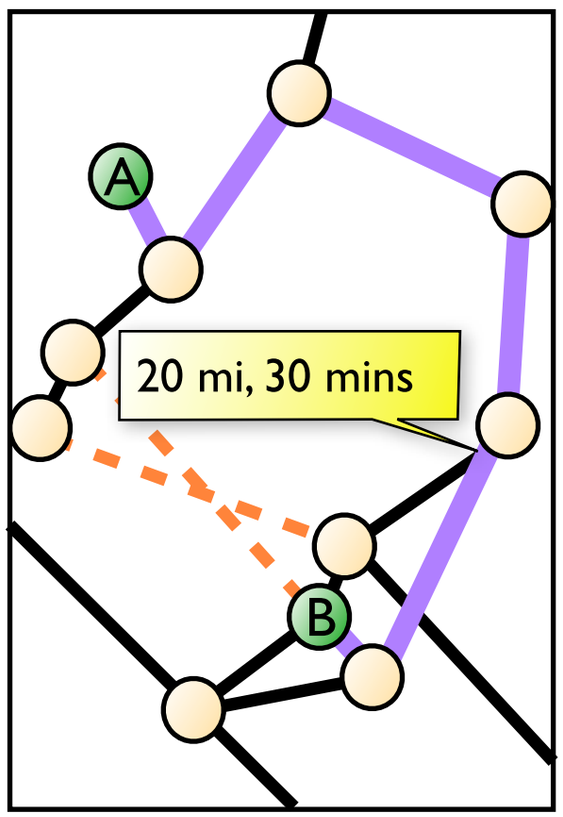
5:00 AM - 6:00 AM



6:00 AM - 7:00 AM

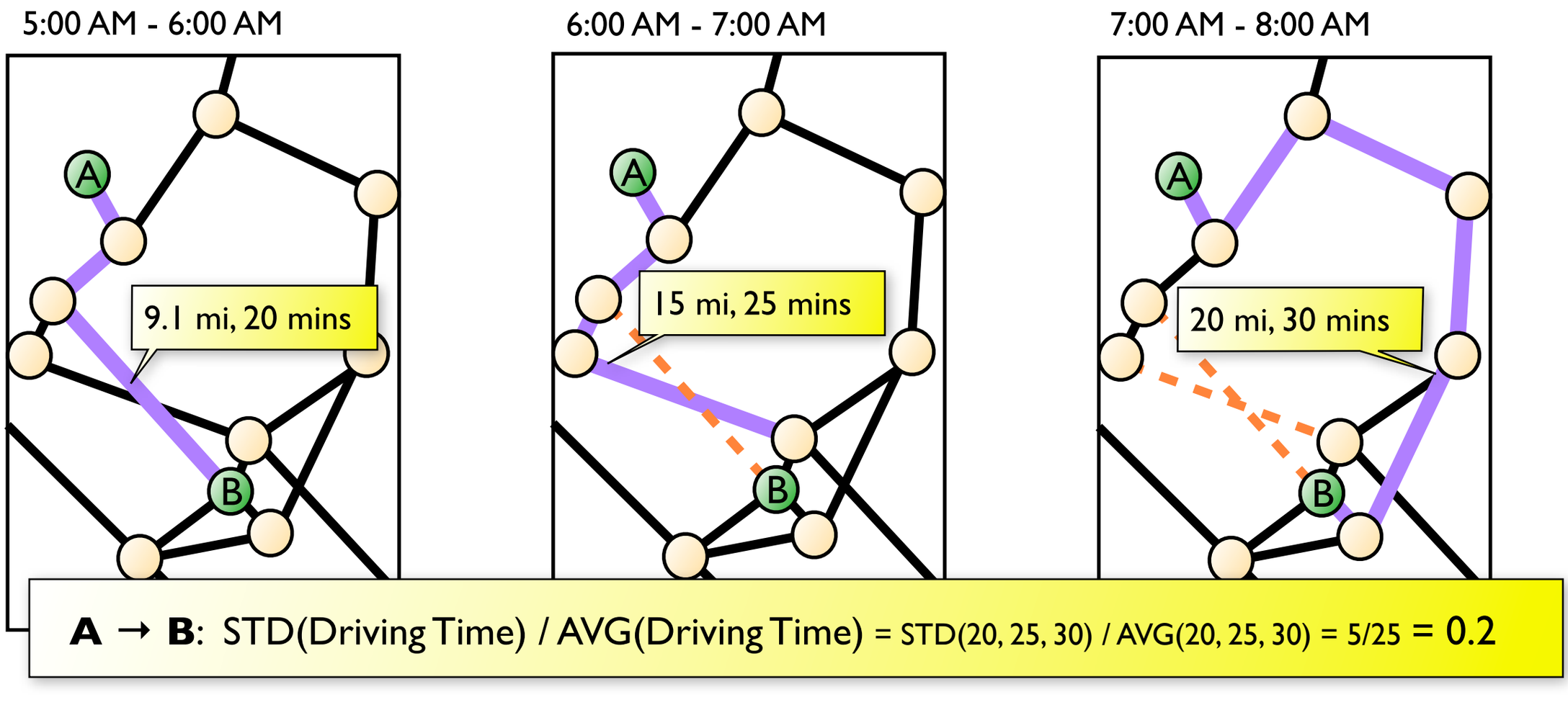


7:00 AM - 8:00 AM



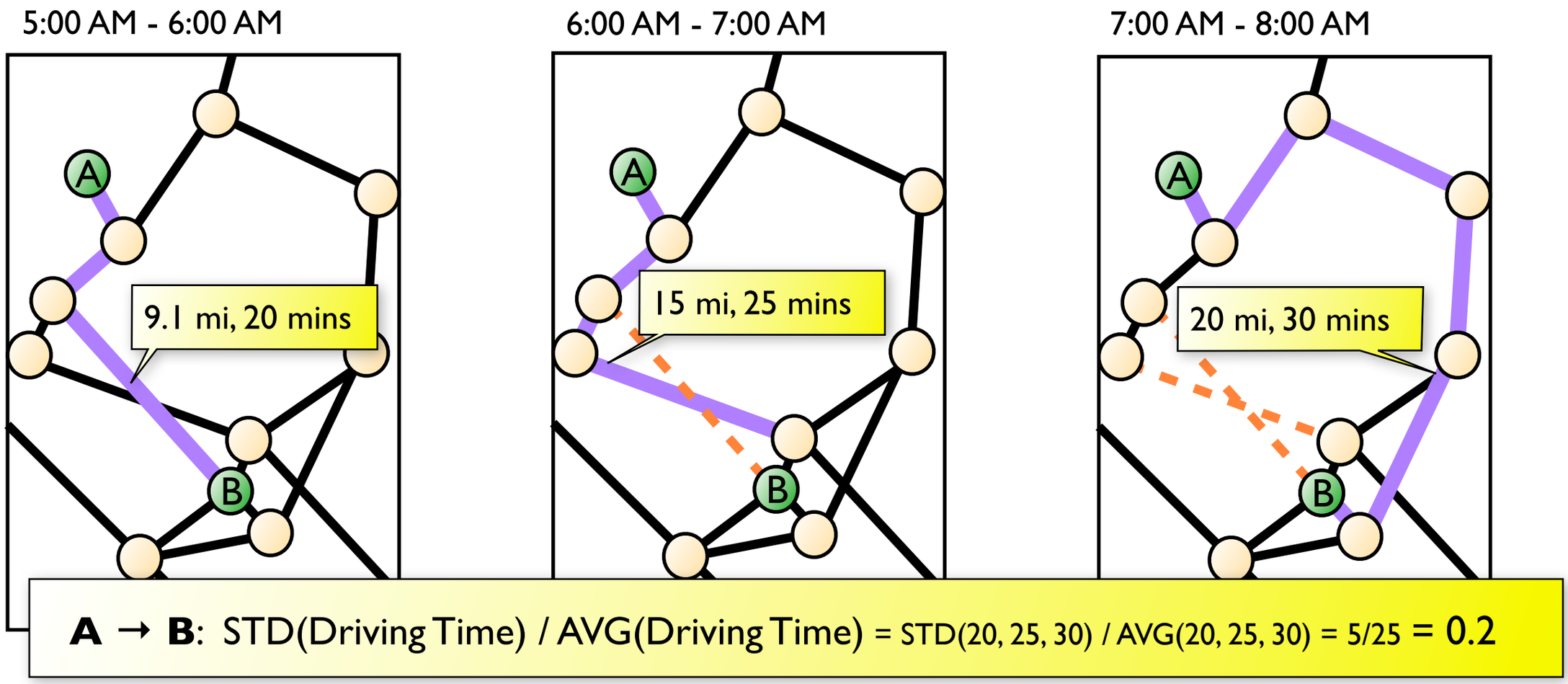
Graph Data Analytics

- Transportation



Graph Data Analytics

- Transportation



- Social and Political Studies / Marketing / National Security
- Epidemic Simulations and Analysis

Limitations of Existing Systems

	inconvenient	inefficient
RDBMSs	no SQL support for graphs	no built-in graph traversal capabilities
Graph Systems	coding for each type of computation	only one graph at a time

Limitations of Existing Systems

	inconvenient	inefficient
RDBMSs	no SQL support for graphs	no built-in graph traversal capabilities
Graph Systems	coding for each type of computation	only one graph at a time

Limitations of Existing Systems

Example: **shortest distance** in **Pregel**

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

```
class MinIntCombiner : public Combiner<int> {
  virtual void Combine(MessageIterator* msgs) {
    int mindist = INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }
};
```

Graph Systems	coding for each type of computation	only one graph at a time

Limitations of Existing Systems

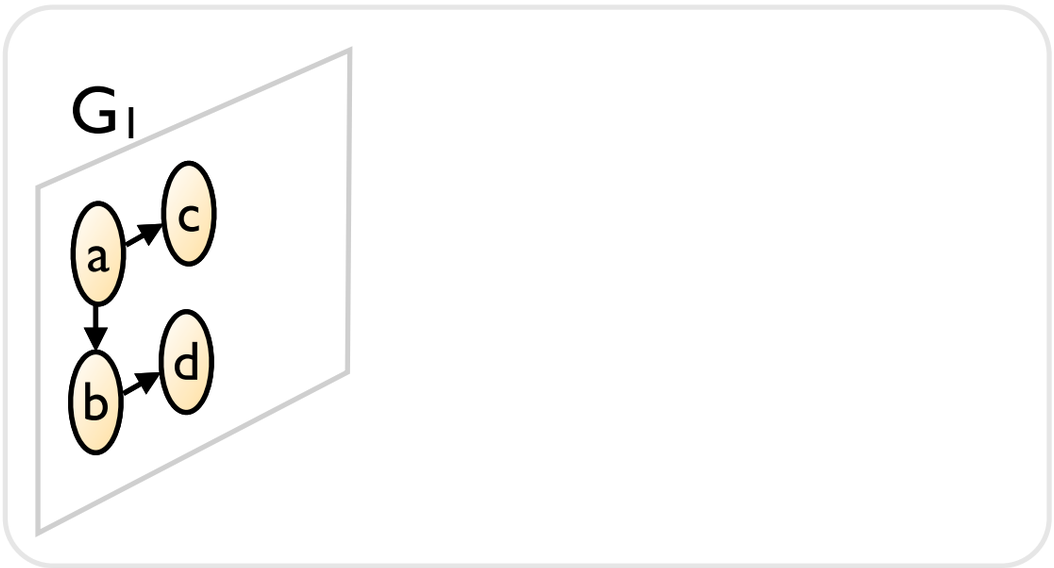
	inconvenient	inefficient
RDBMSs	no SQL support for graphs	no built-in graph traversal capabilities
Graph Systems	coding for each type of computation	only one graph at a time

The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs

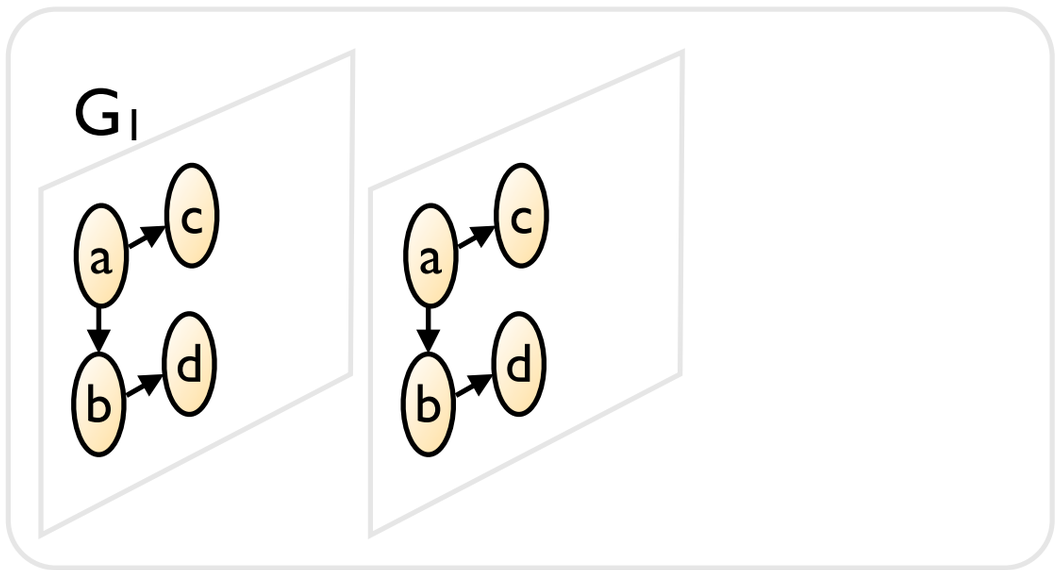
The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



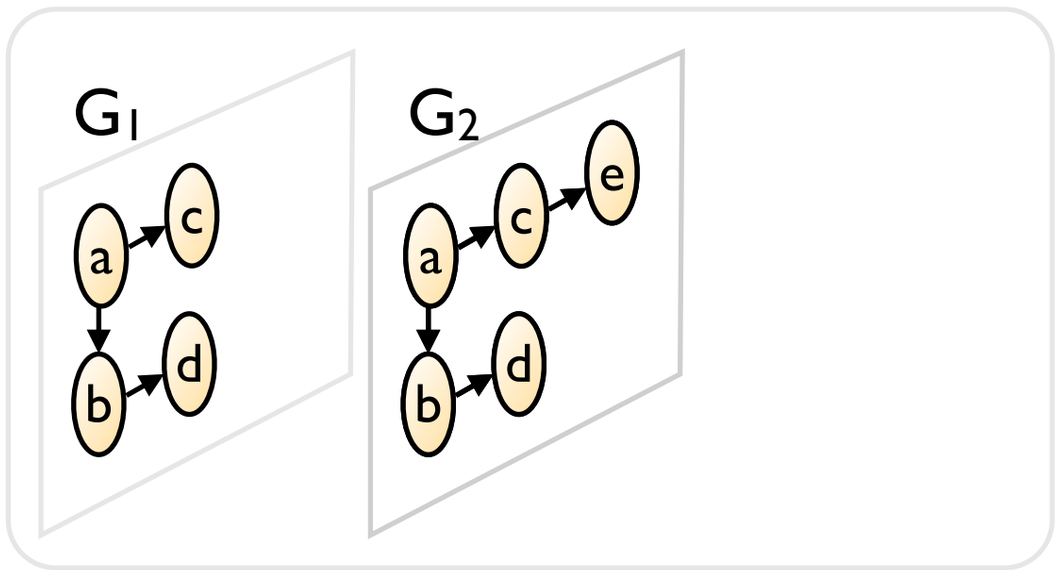
The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



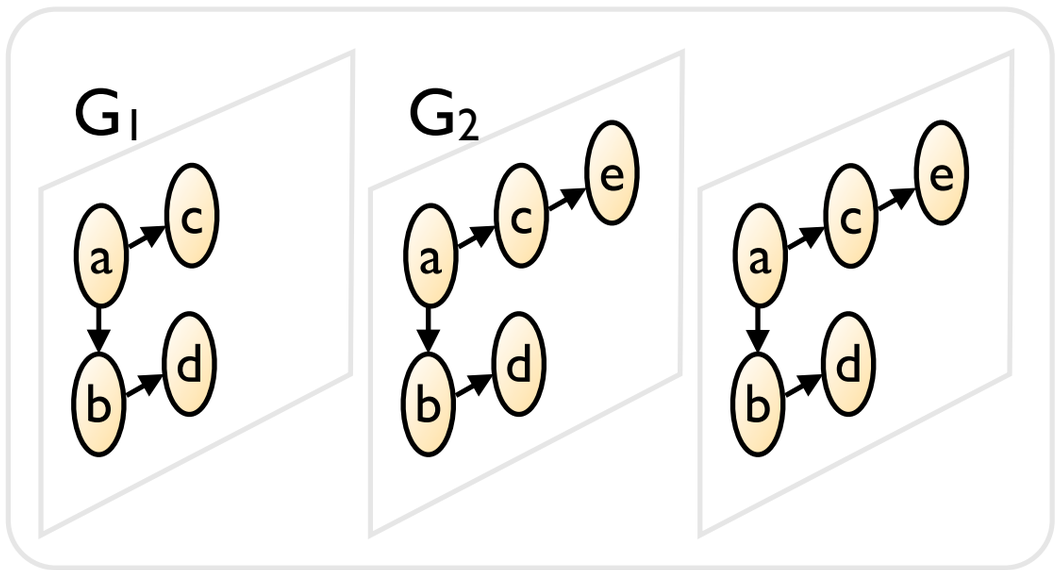
The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



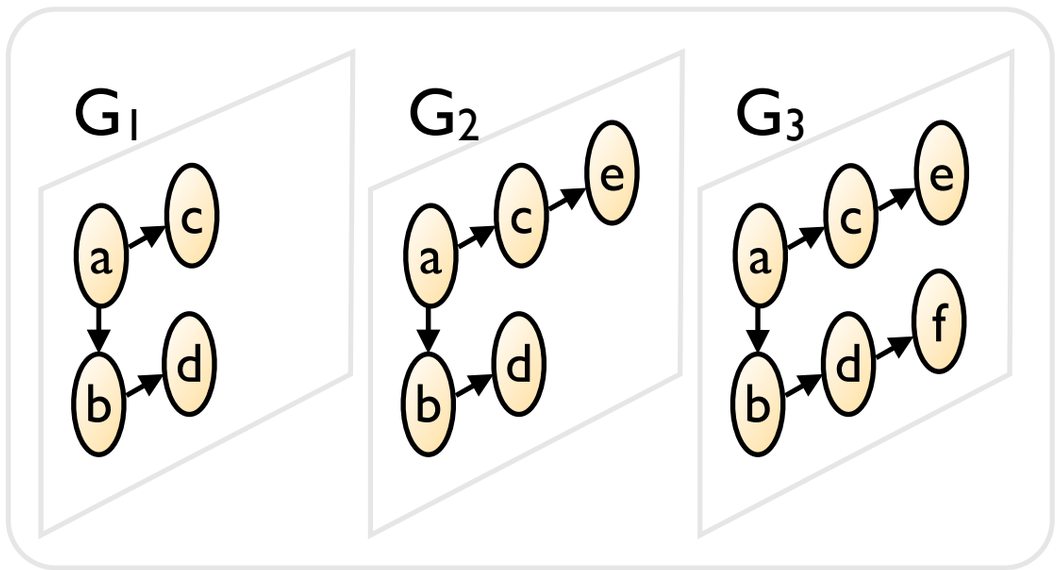
The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



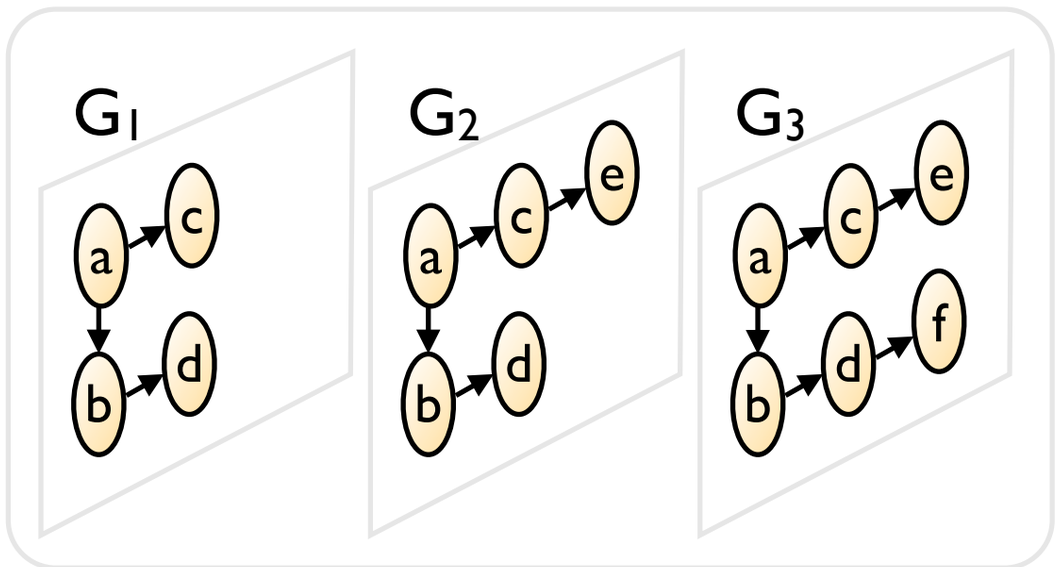
The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



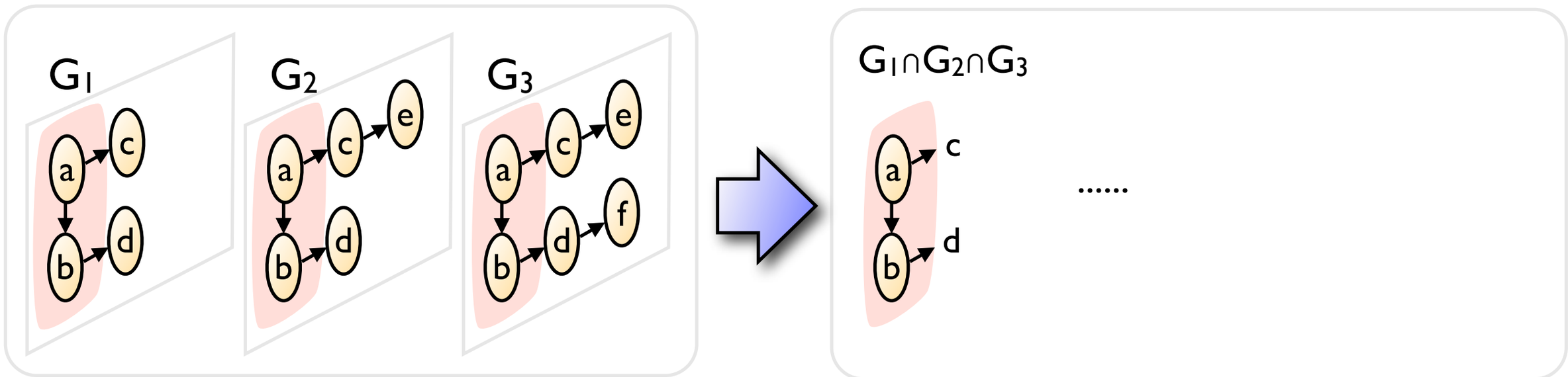
The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



The G* Project

- Goal
  - convenient and efficient queries on graphs using a server cluster
- Key Features
  - succinct graph query language
  - distributed, deduplicate storage of graphs
  - parallel query processing that shares computations across graphs



Distribution of Graph Data

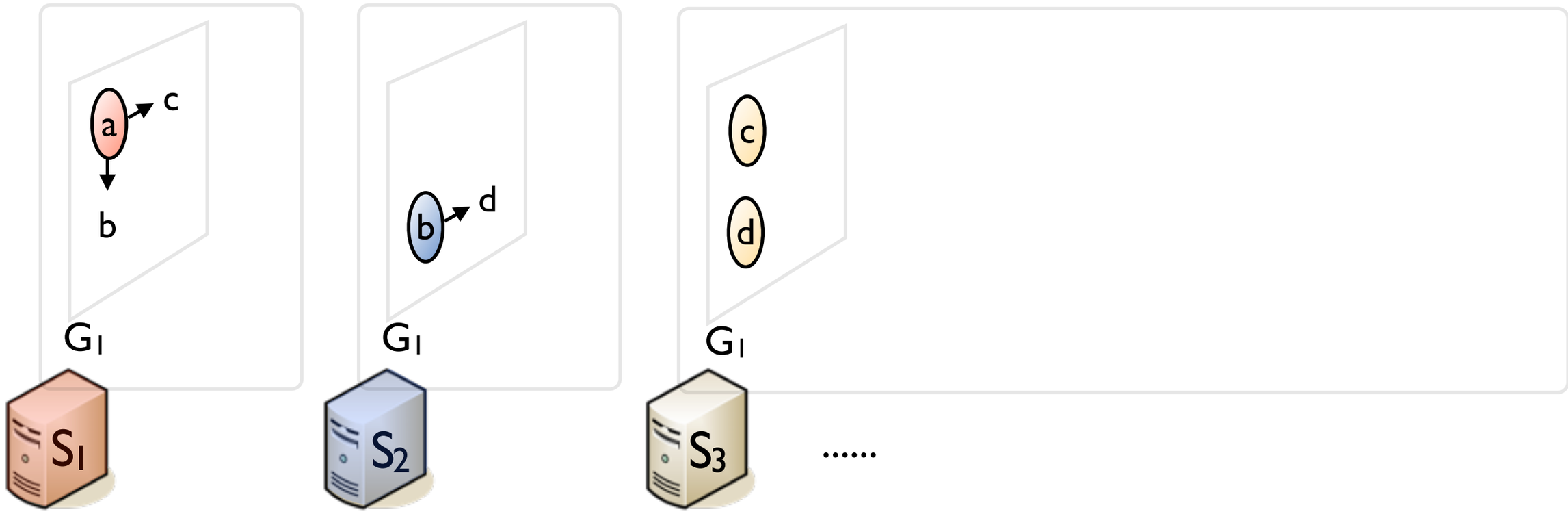
Distribution of Graph Data

.....

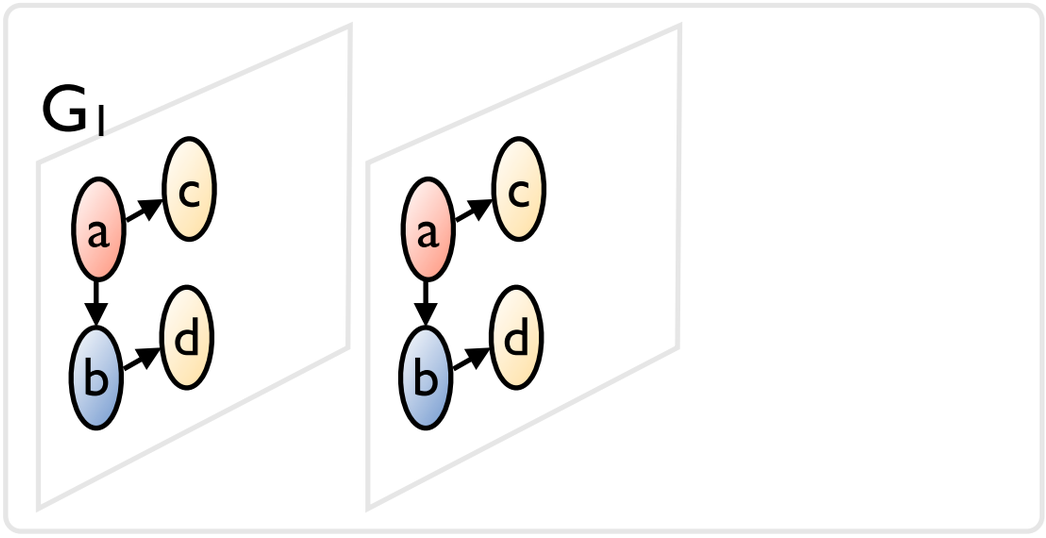
Distribution of Graph Data

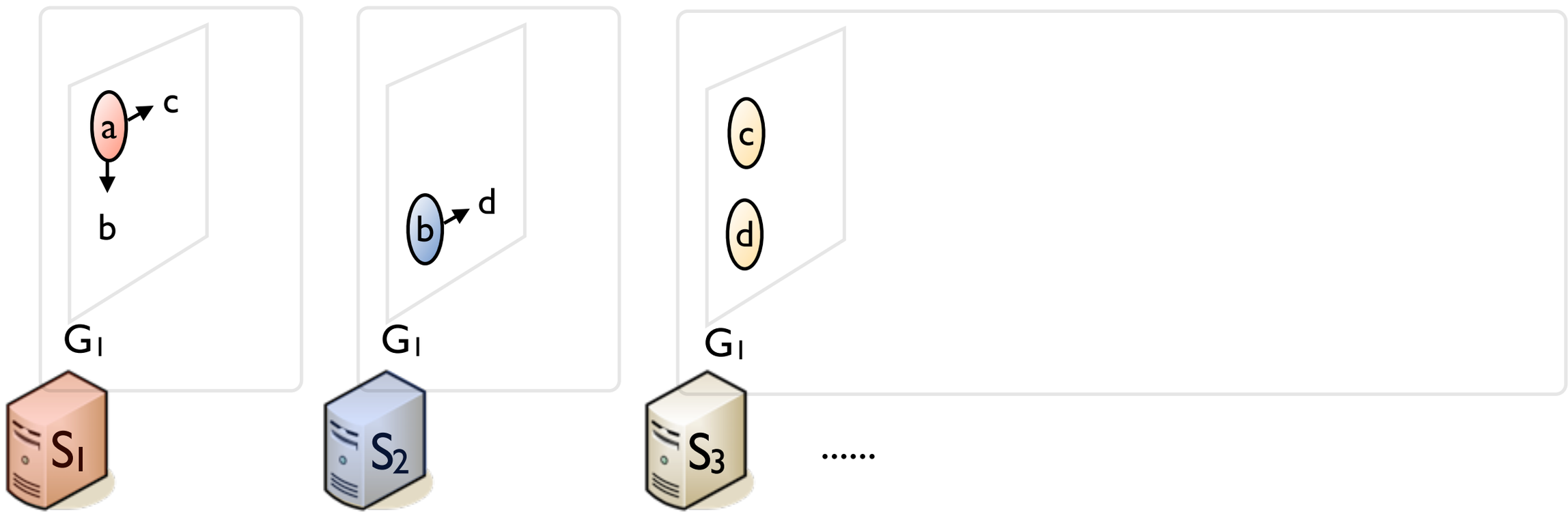
.....

Distribution of Graph Data

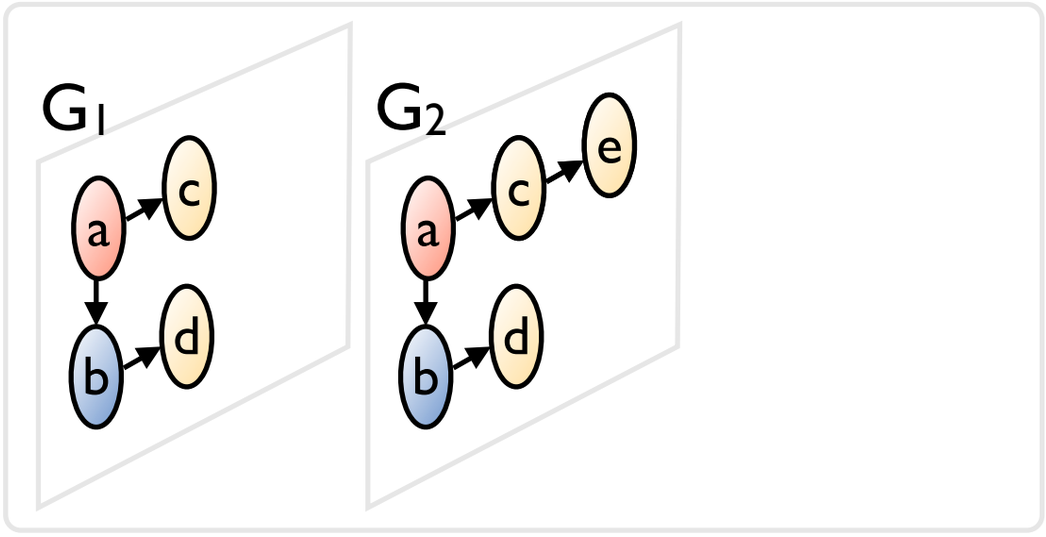


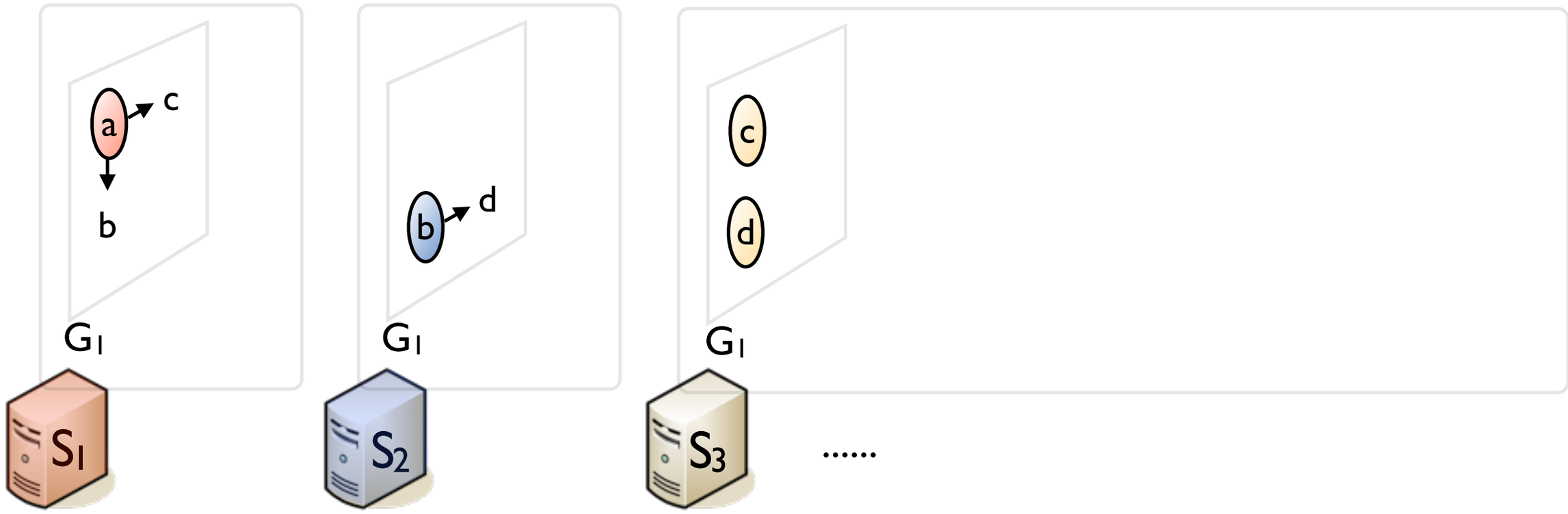
Distribution of Graph Data



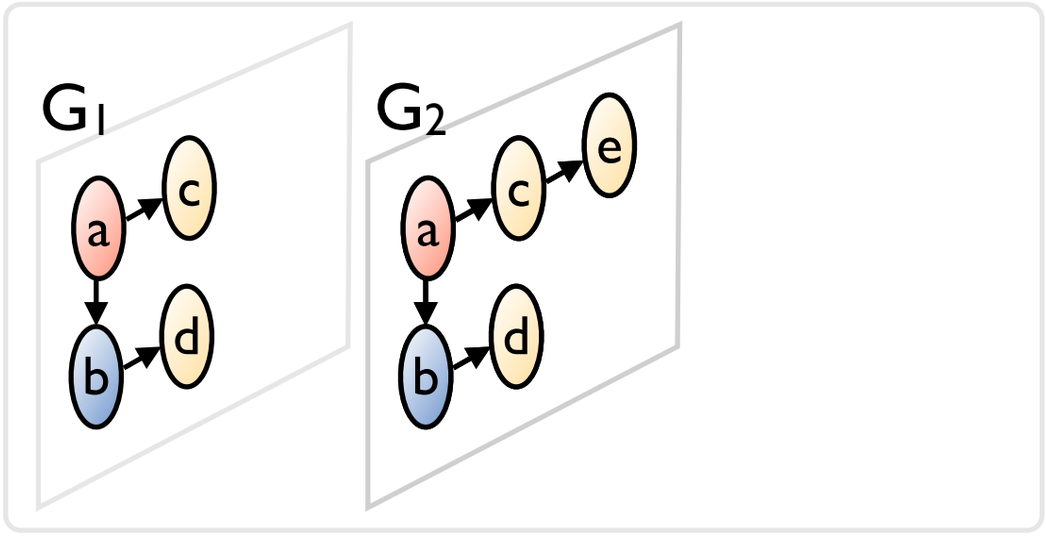


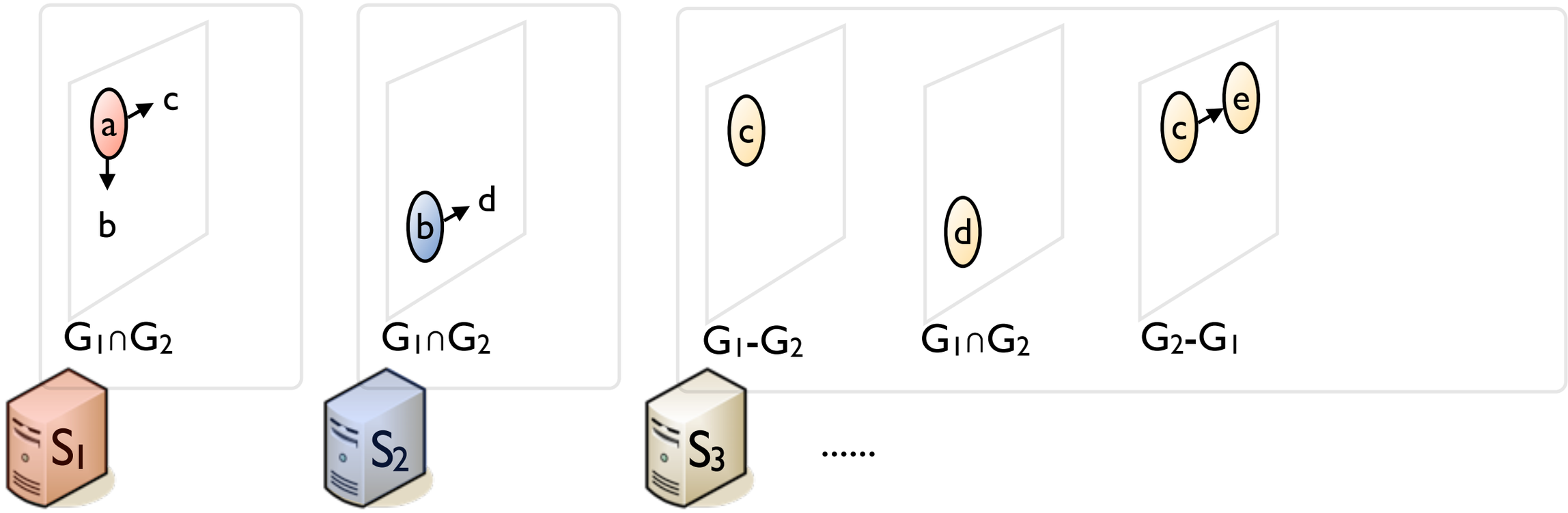
Distribution of Graph Data



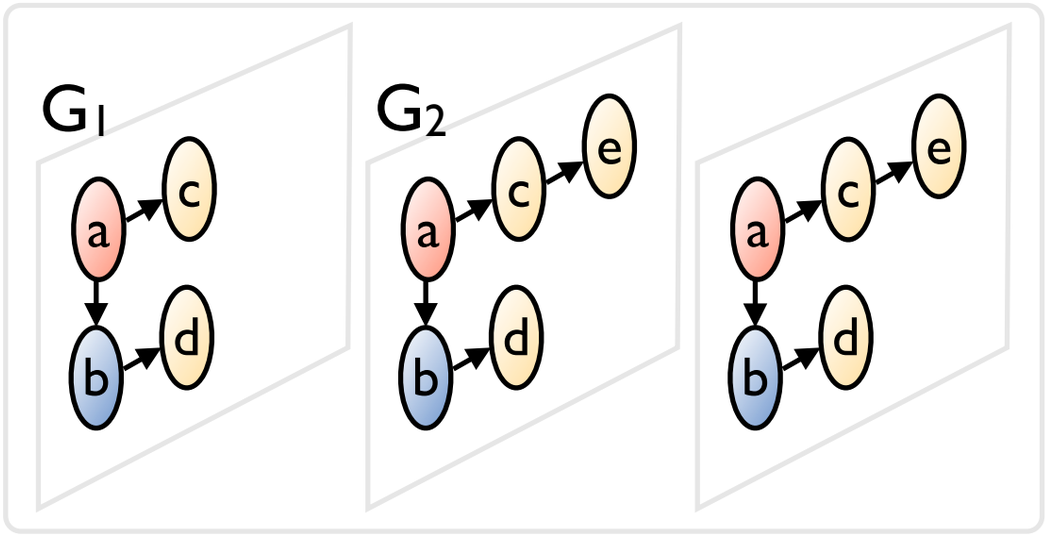


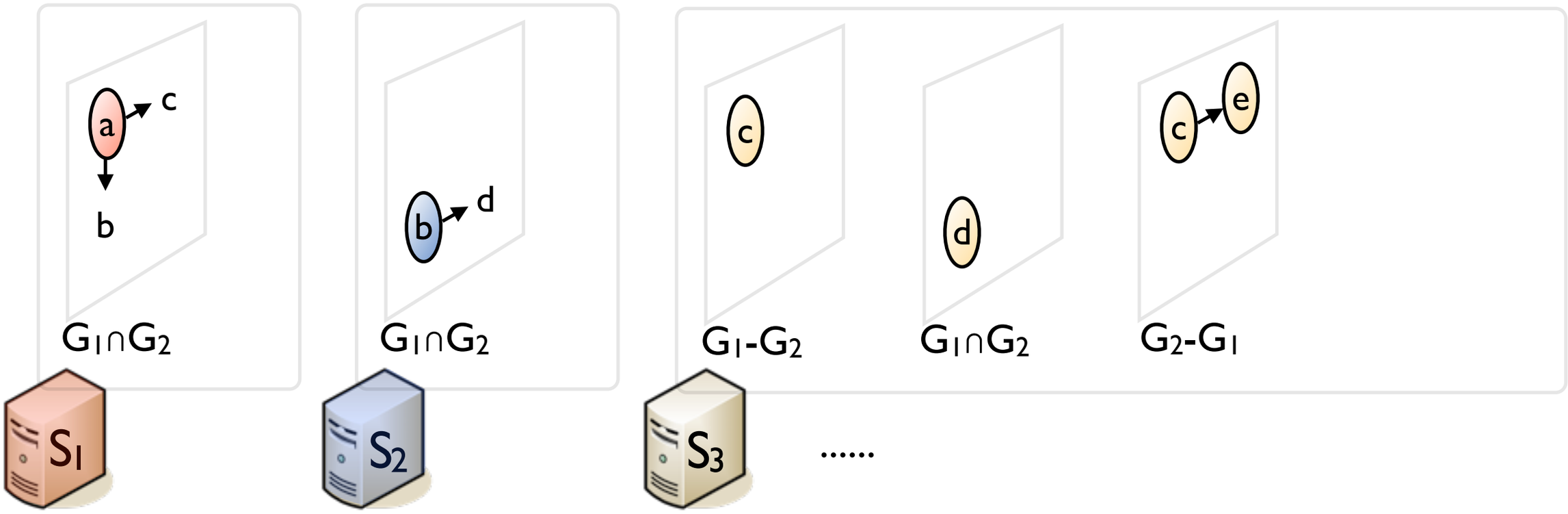
Distribution of Graph Data



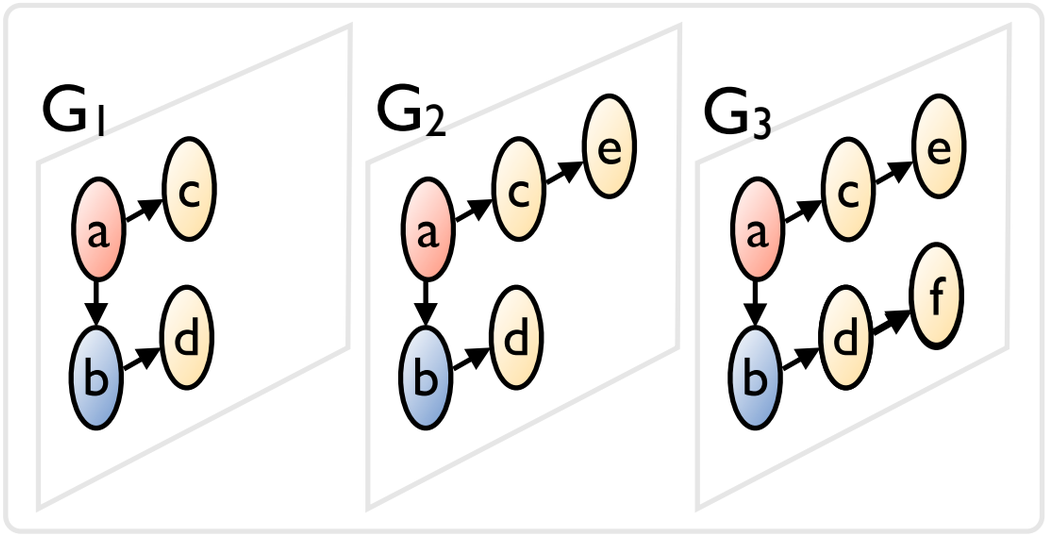


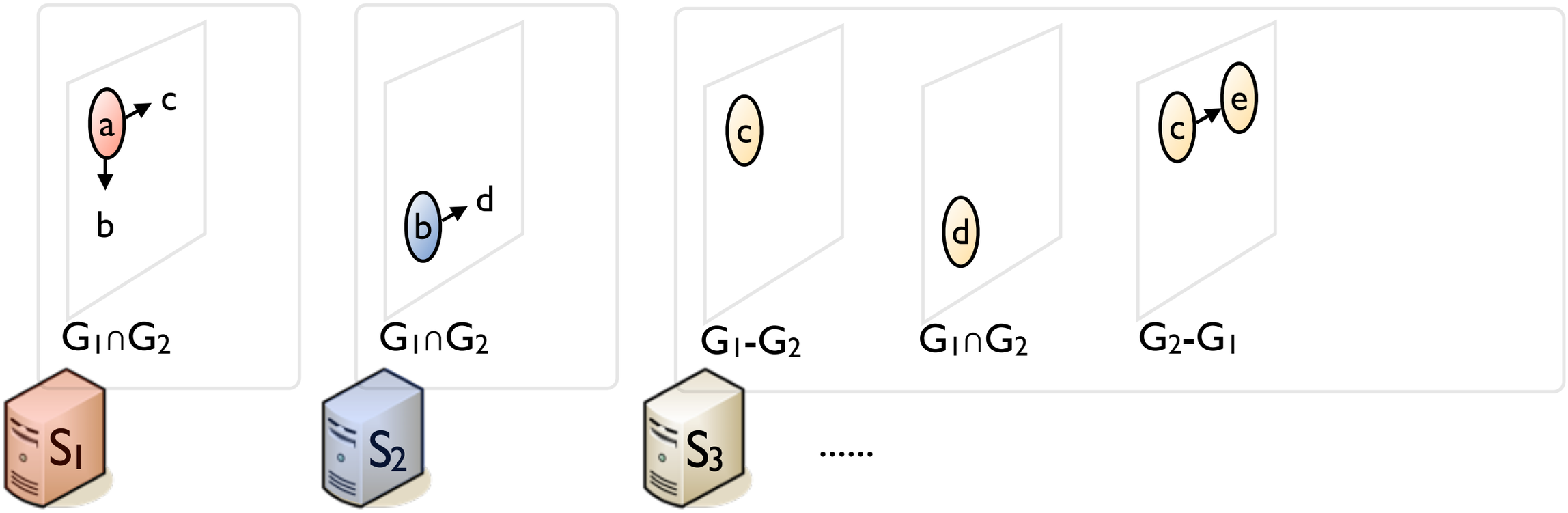
Distribution of Graph Data



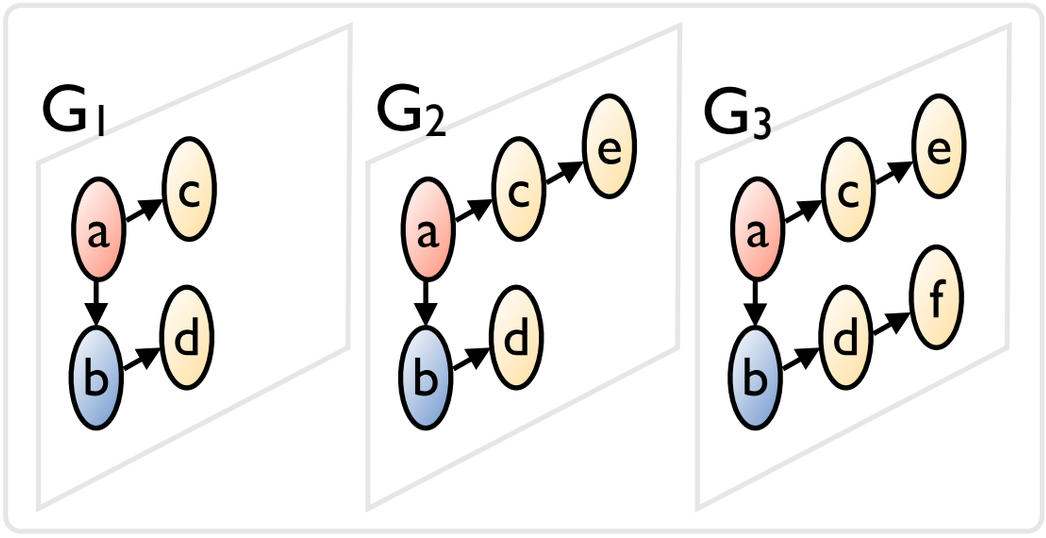


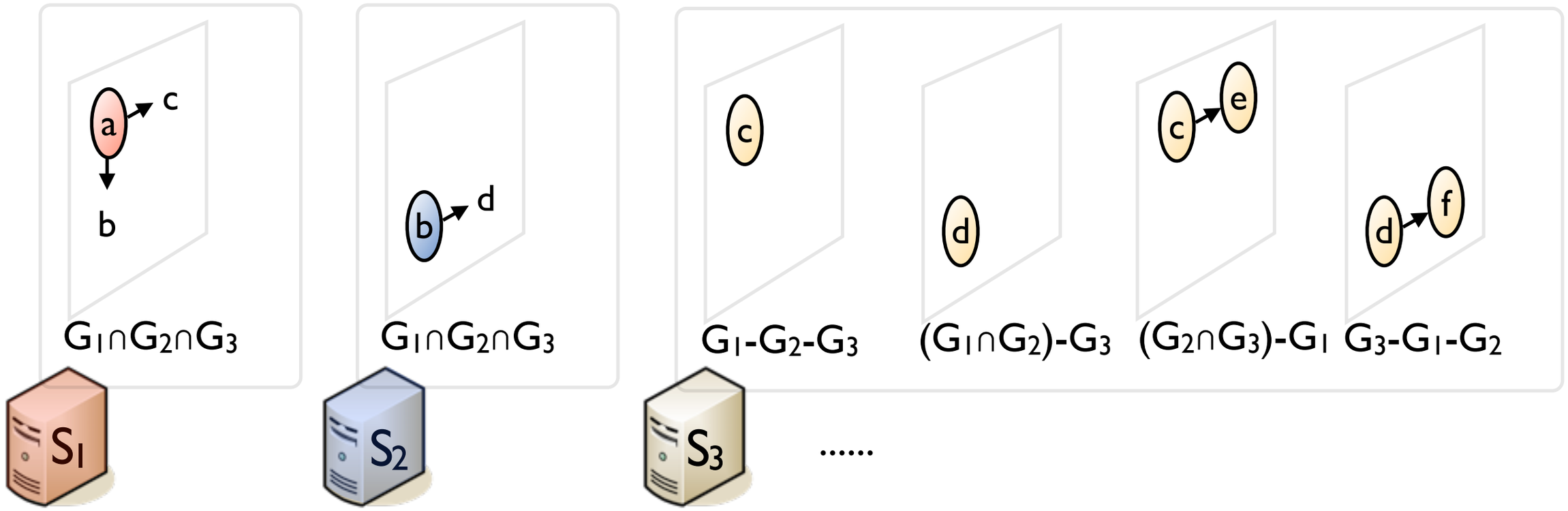
Distribution of Graph Data



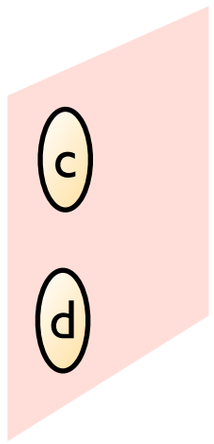


Distribution of Graph Data



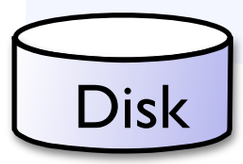


Storage of Graphs (on server)

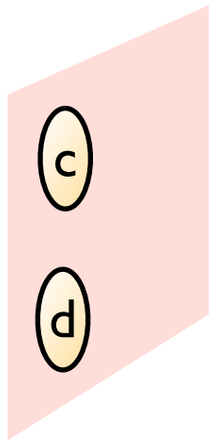


G_1

Memory



Storage of Graphs (on server)

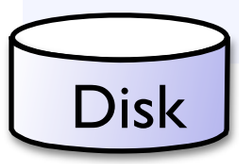


G_1

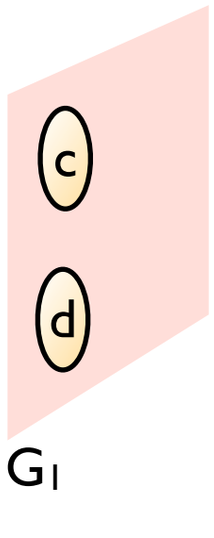
[1] attributes of vertex c

[2] attributes of vertex d

Memory



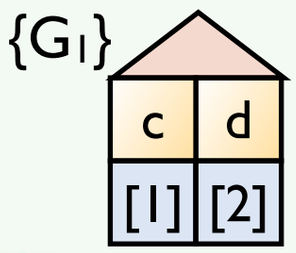
Storage of Graphs (on server)



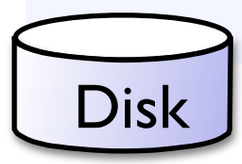
G_1

[1] attributes of vertex c

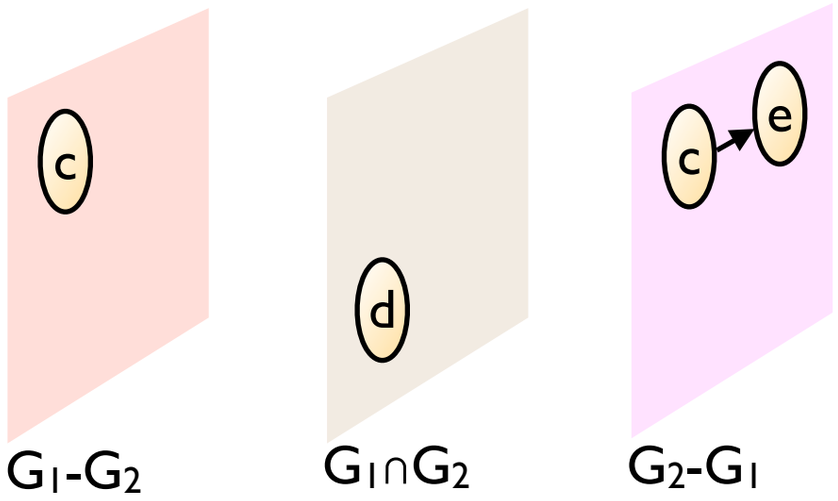
[2] attributes of vertex d



Memory

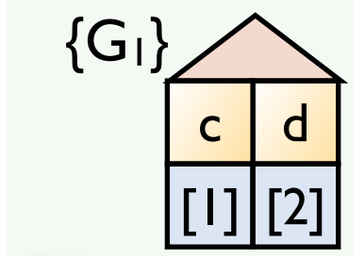


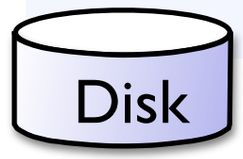
Storage of Graphs (on server)



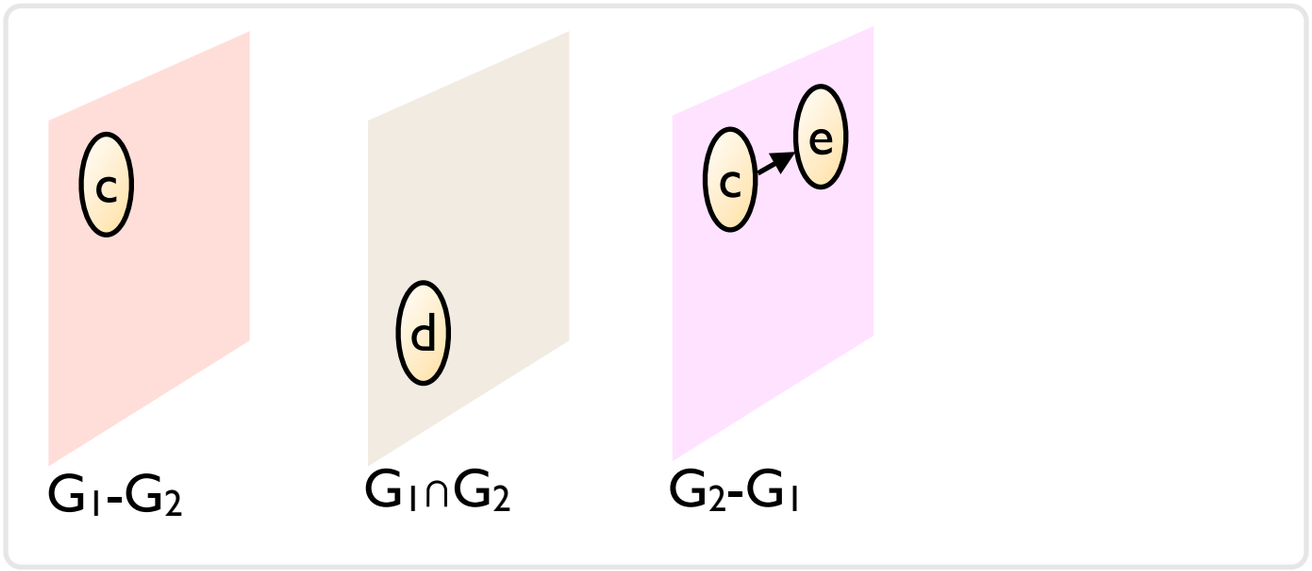
[1] attributes of vertex c

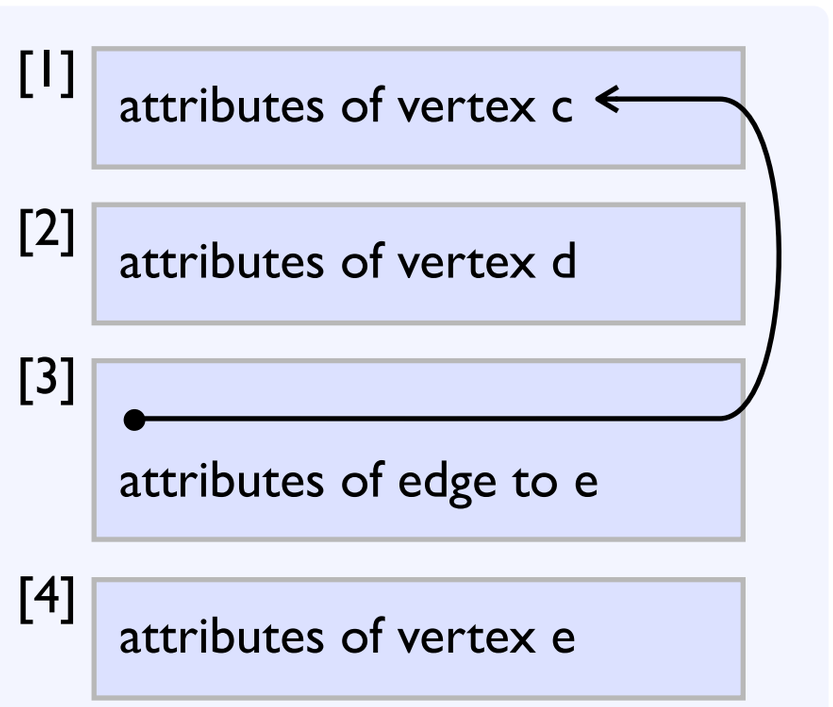
[2] attributes of vertex d

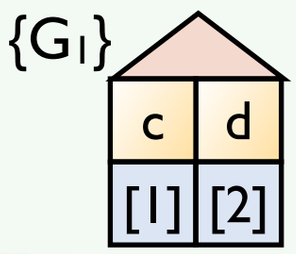




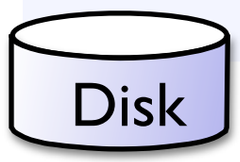
Storage of Graphs (on server)



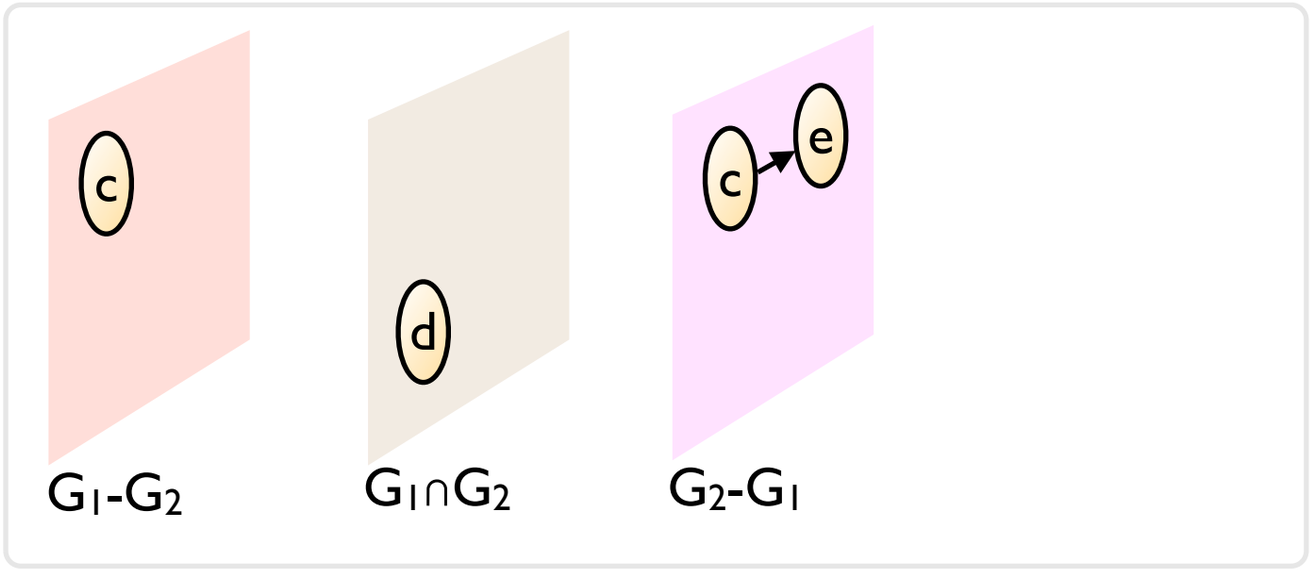


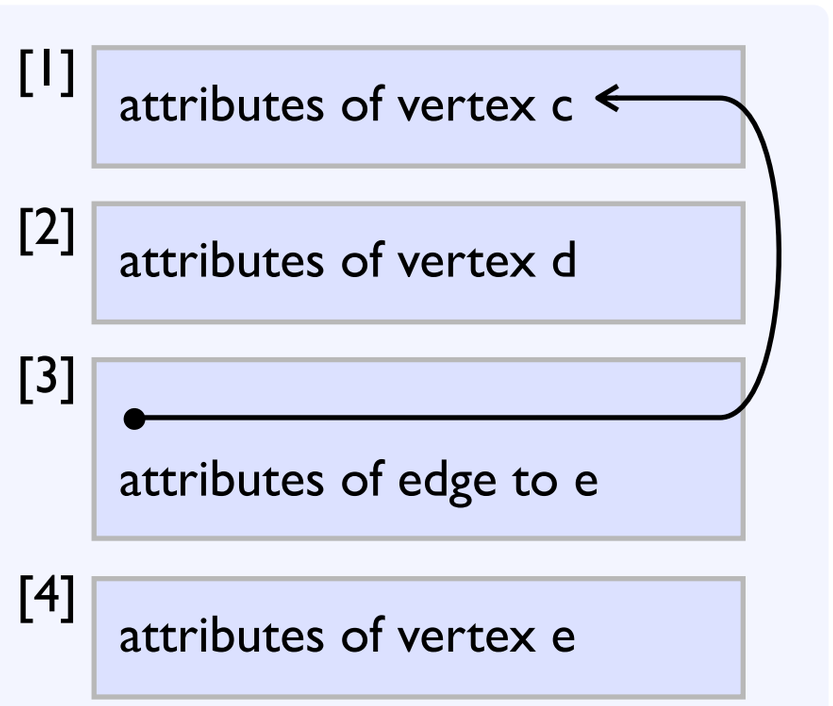


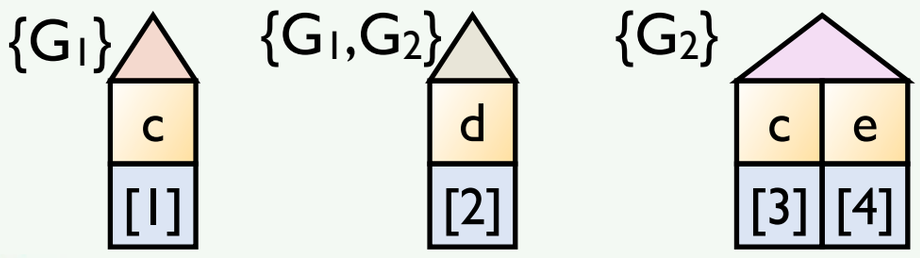
Memory



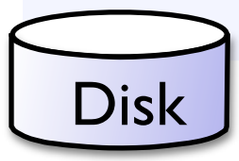
Storage of Graphs (on server)





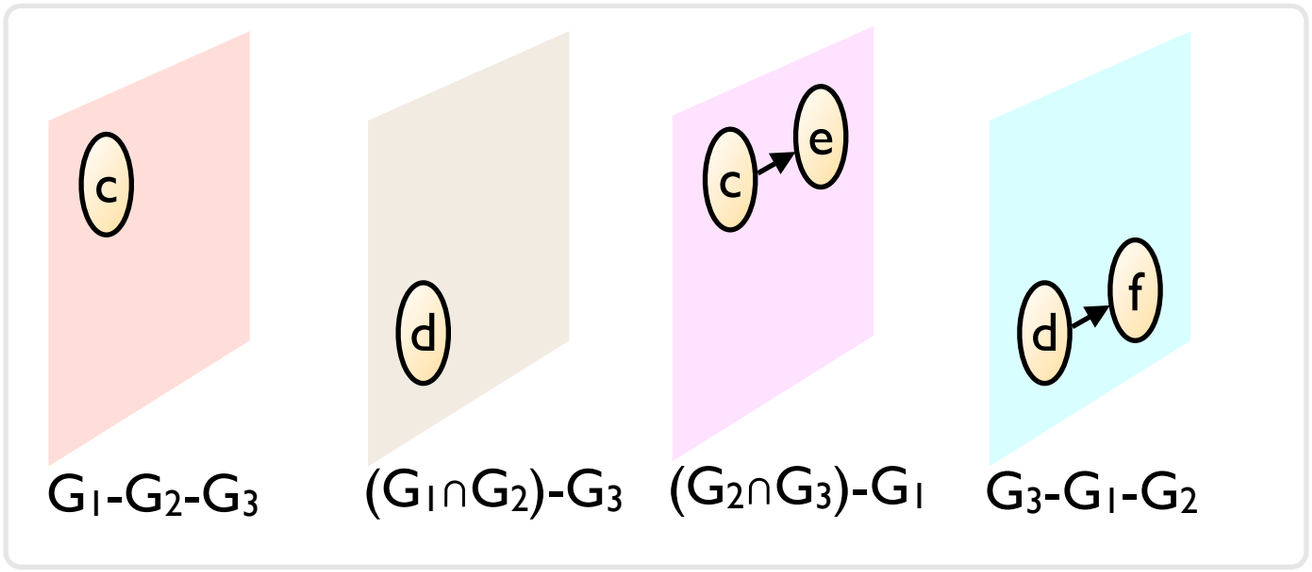


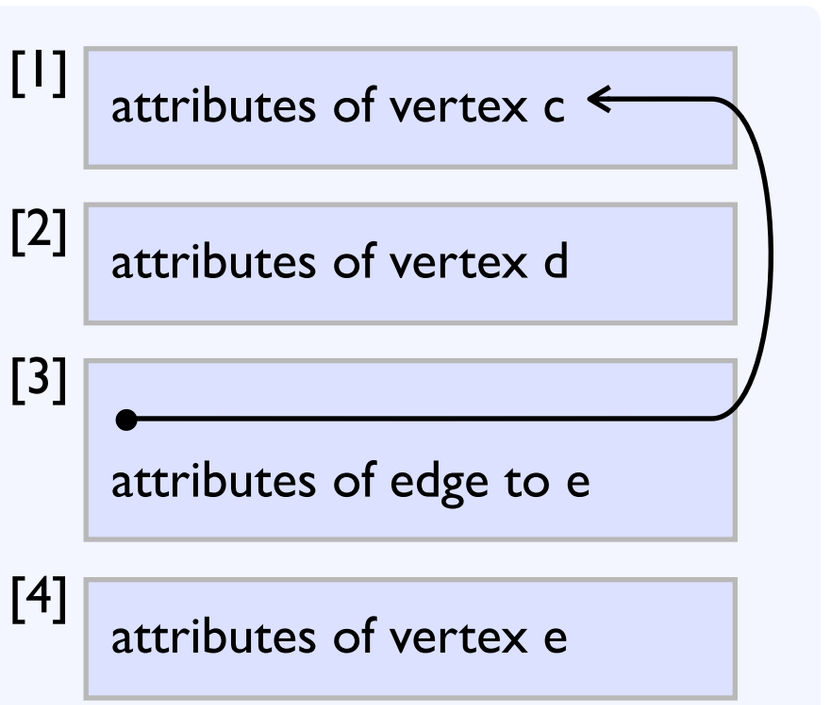
Memory

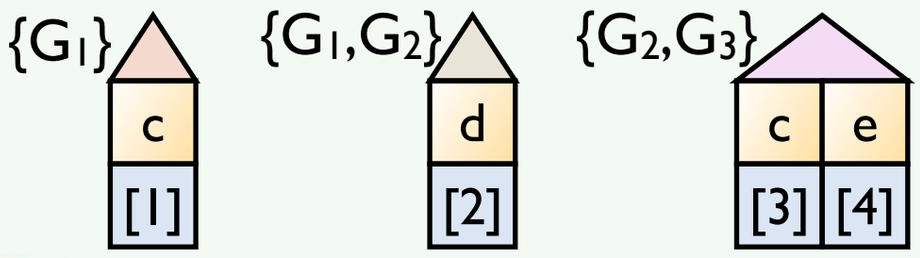


Disk

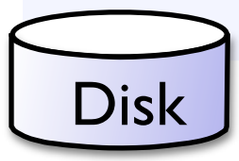
Storage of Graphs (on server)



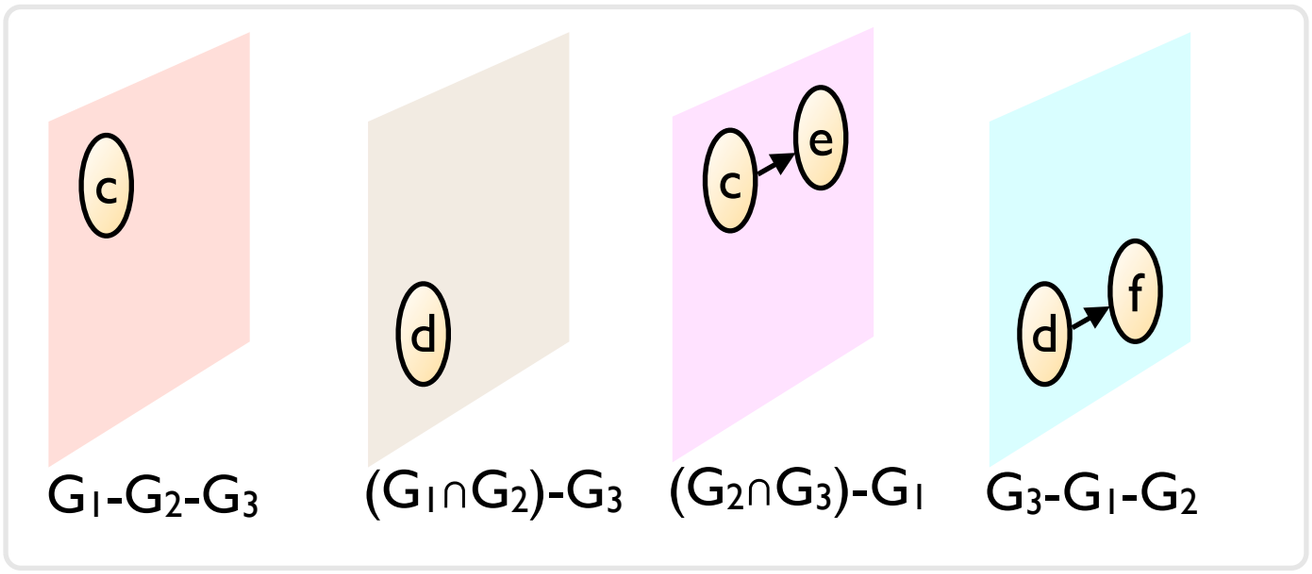


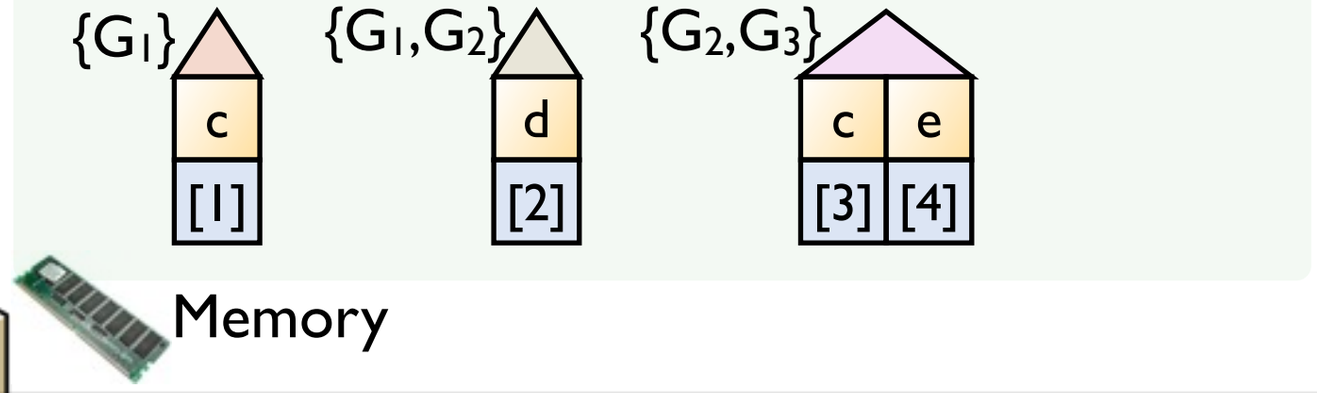


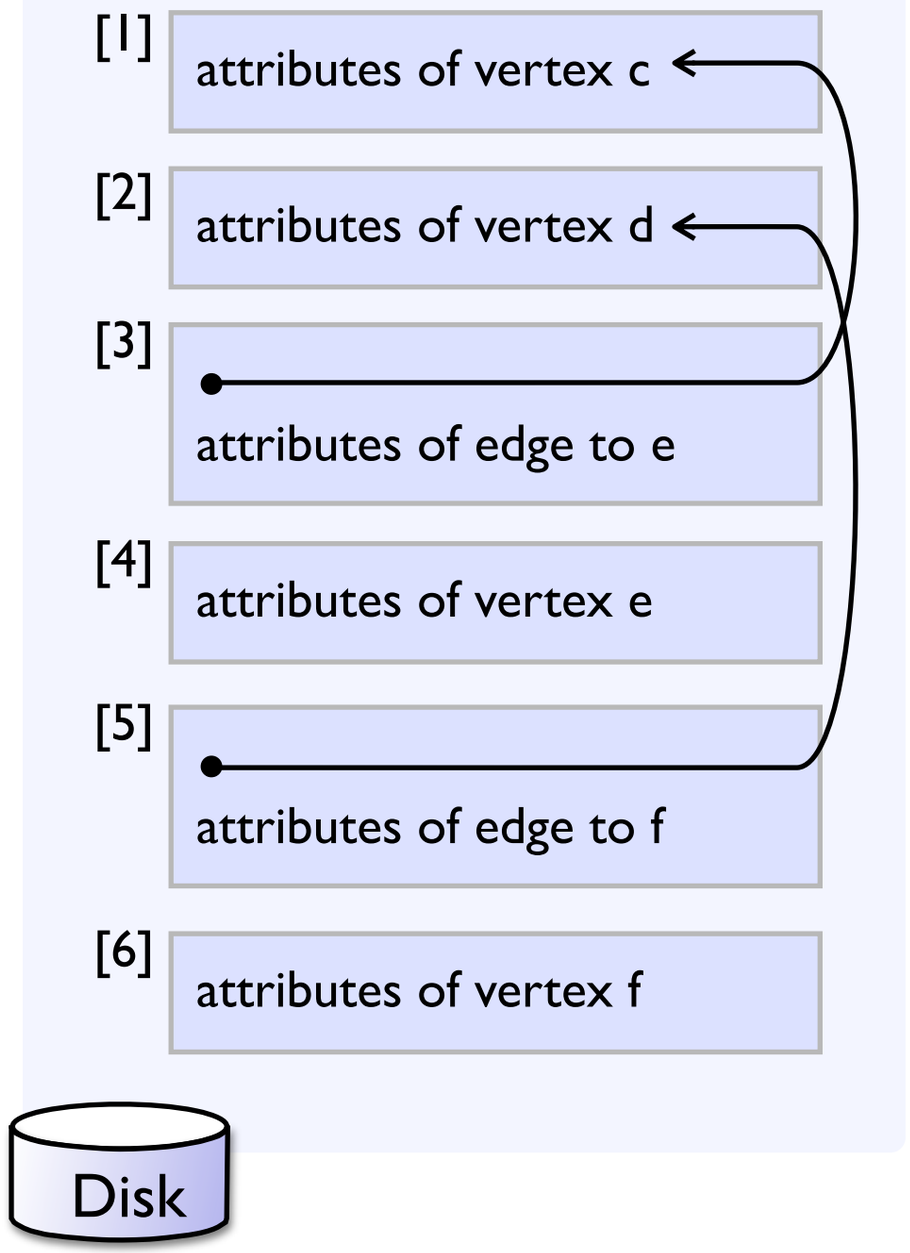
Memory



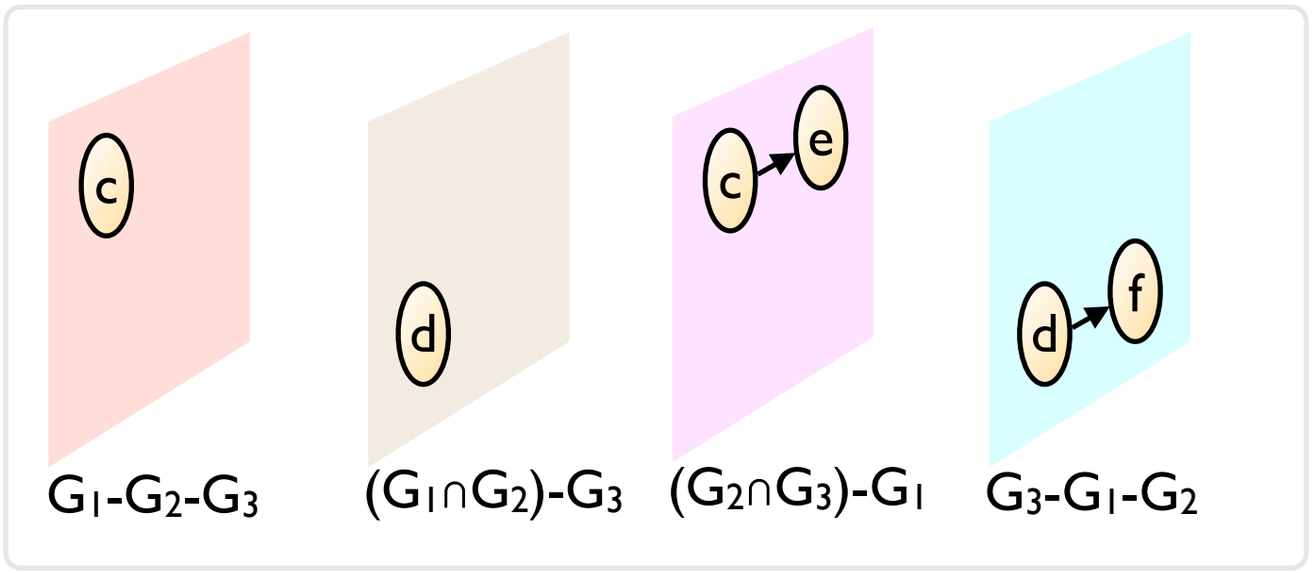
Storage of Graphs (on server)

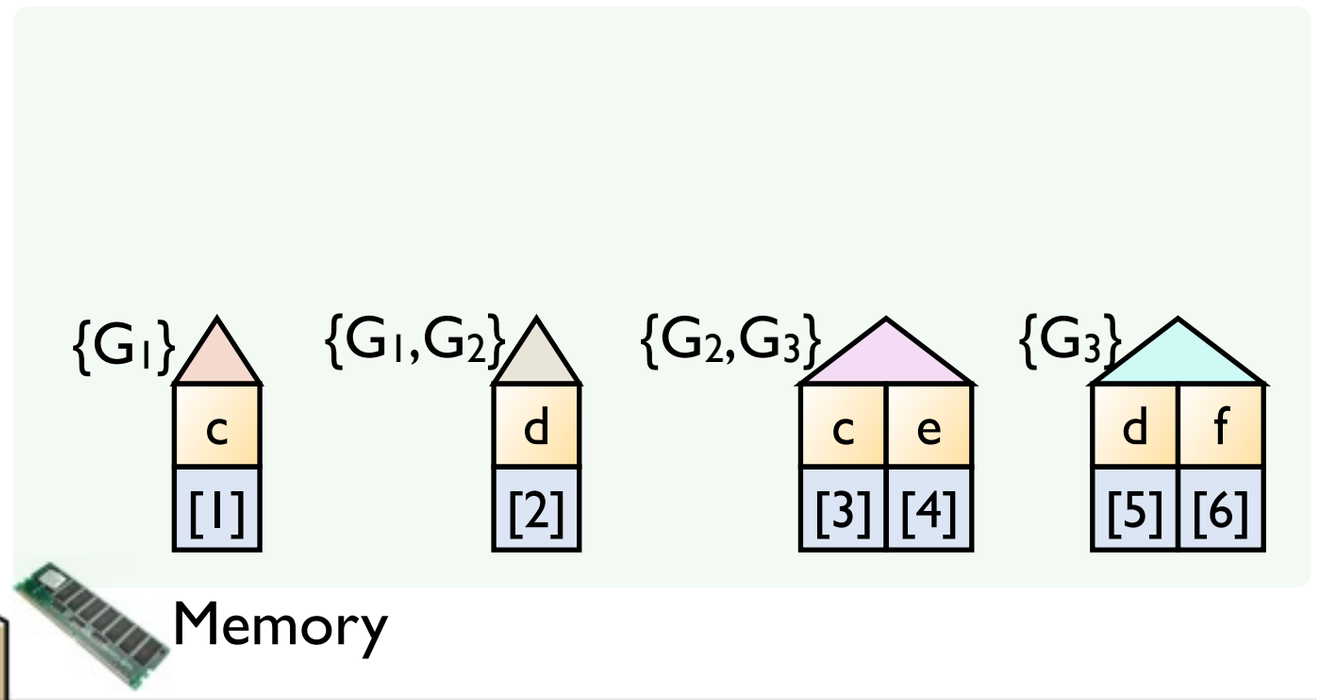


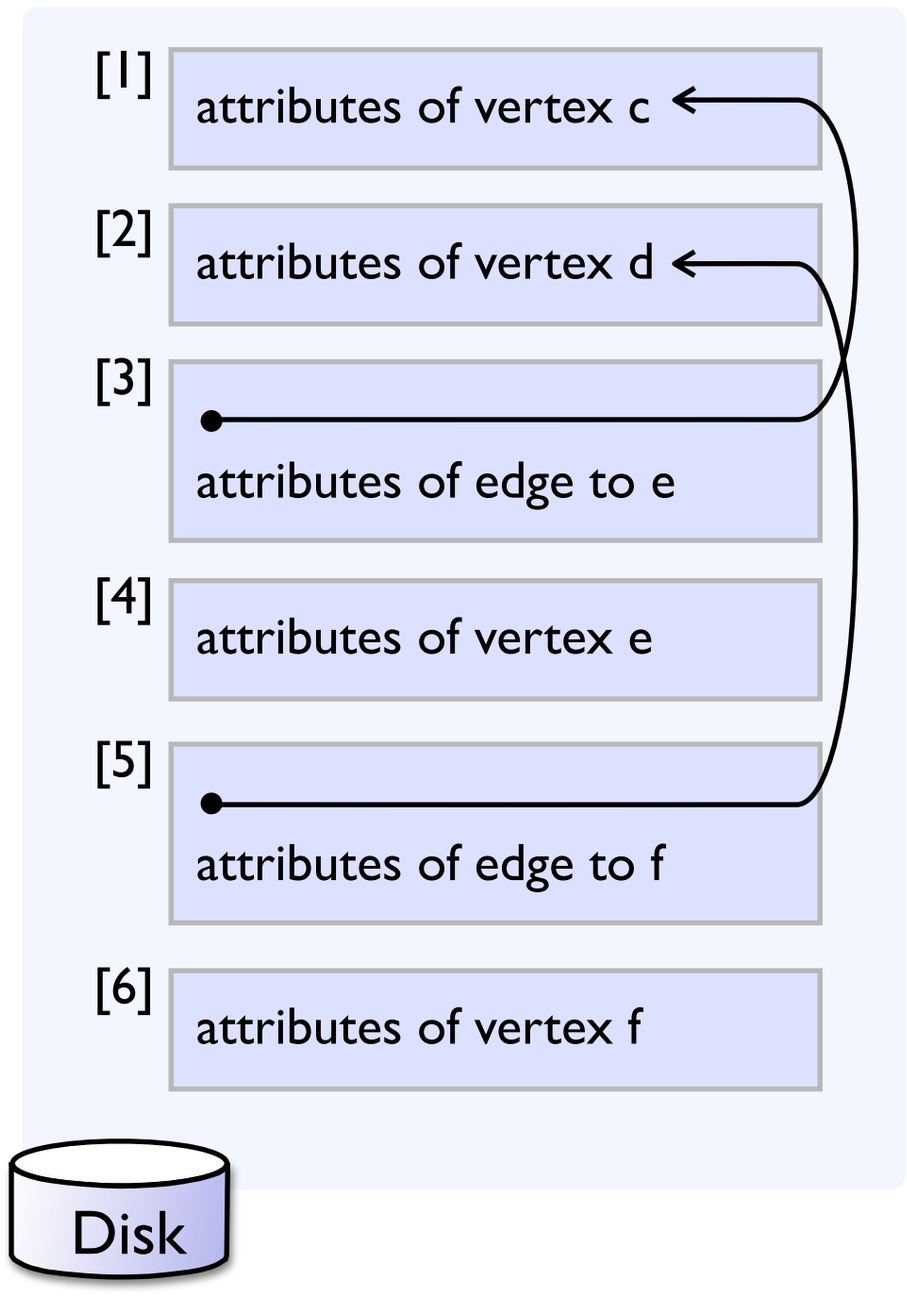




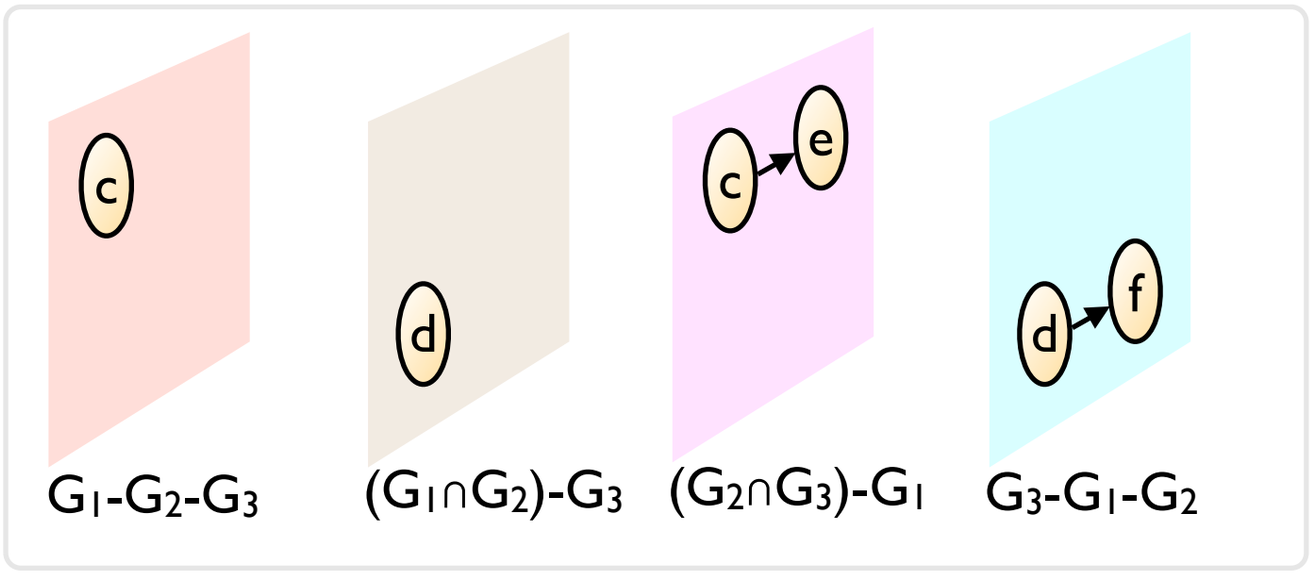
Storage of Graphs (on server)

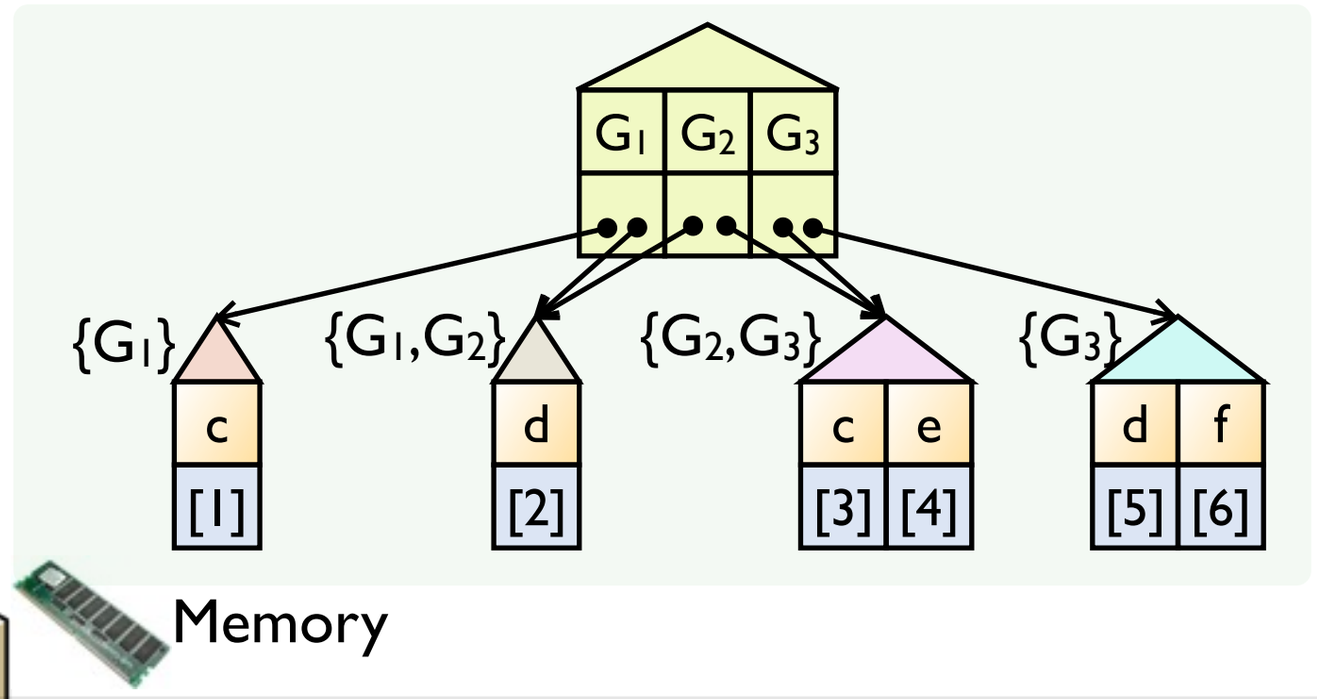


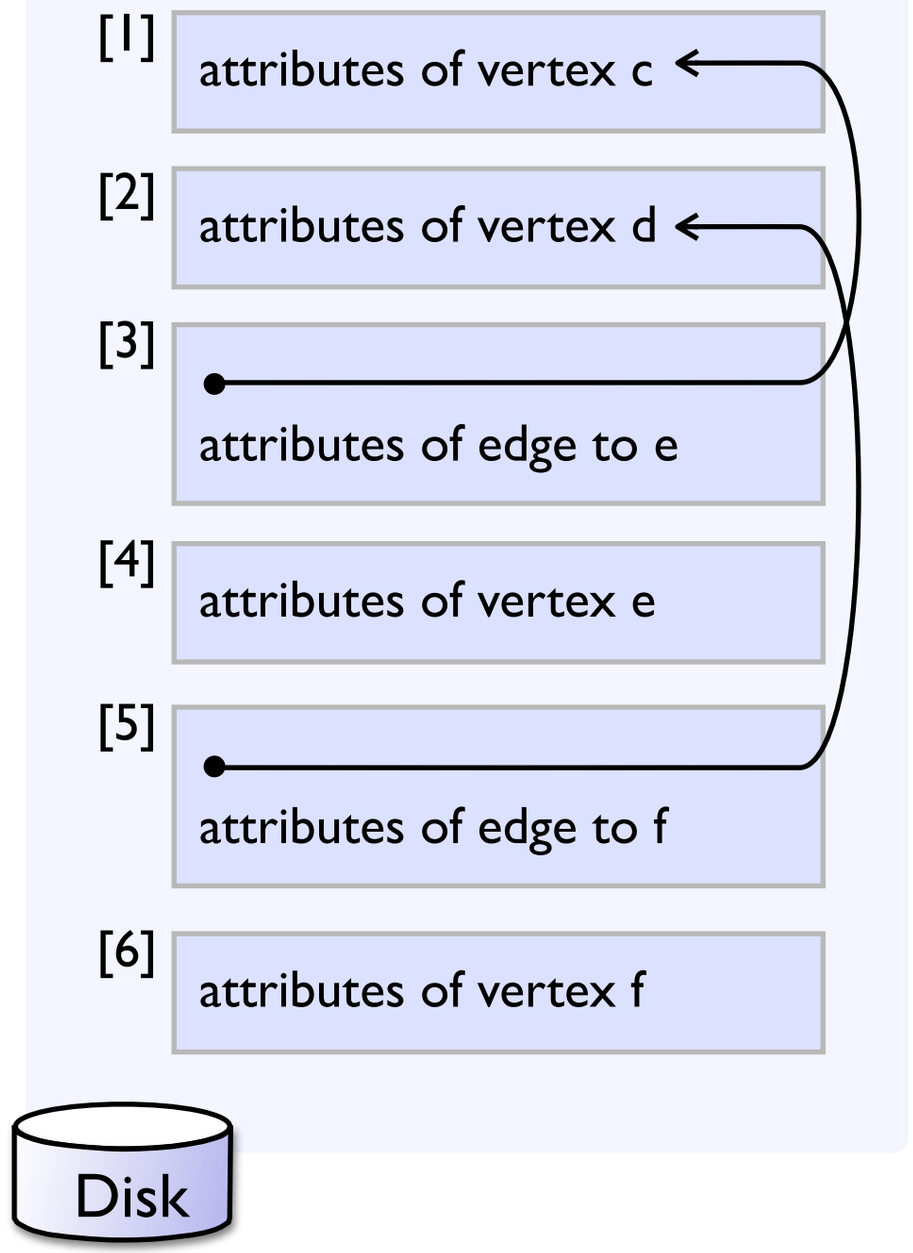




Storage of Graphs (on server)

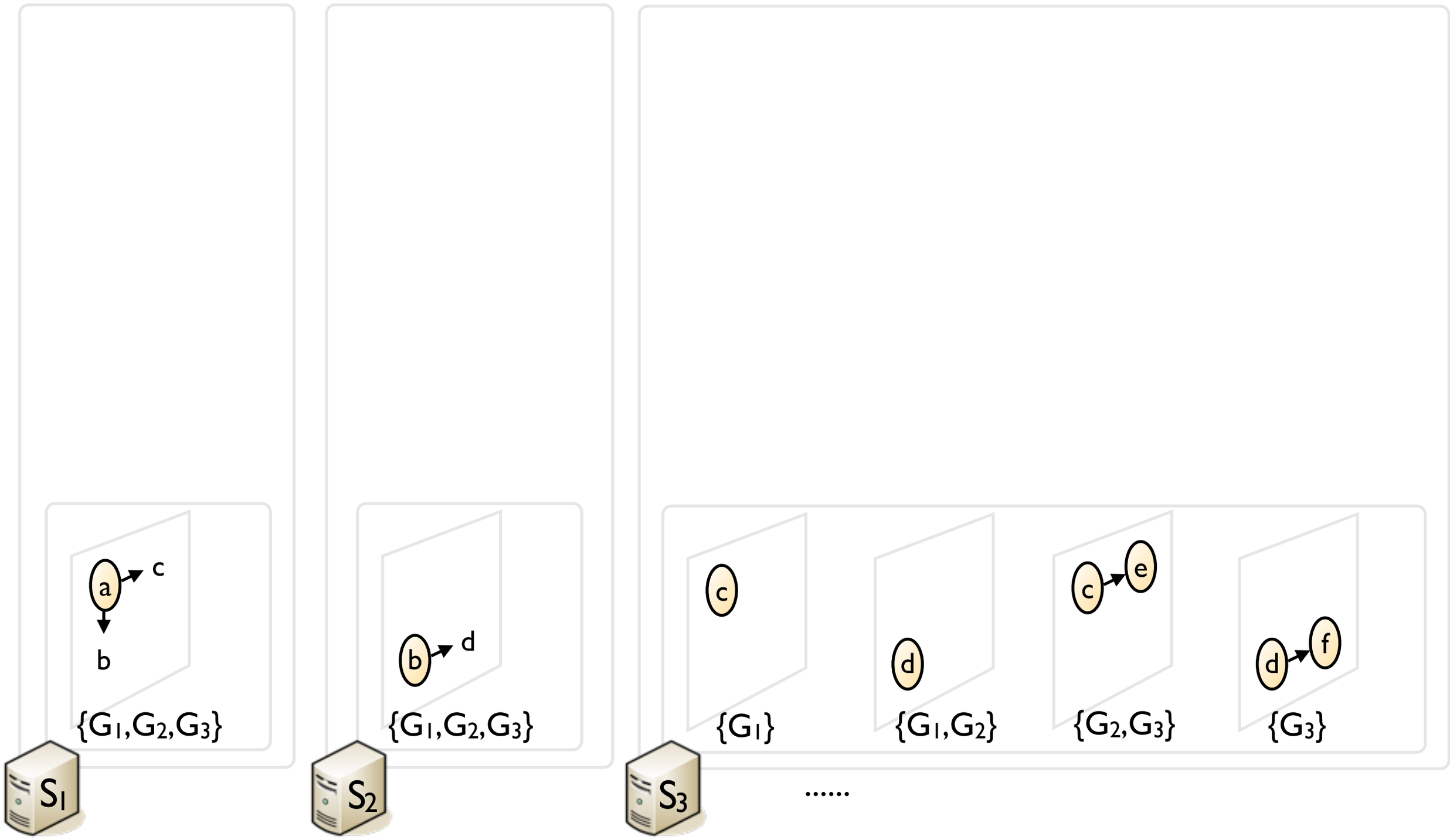






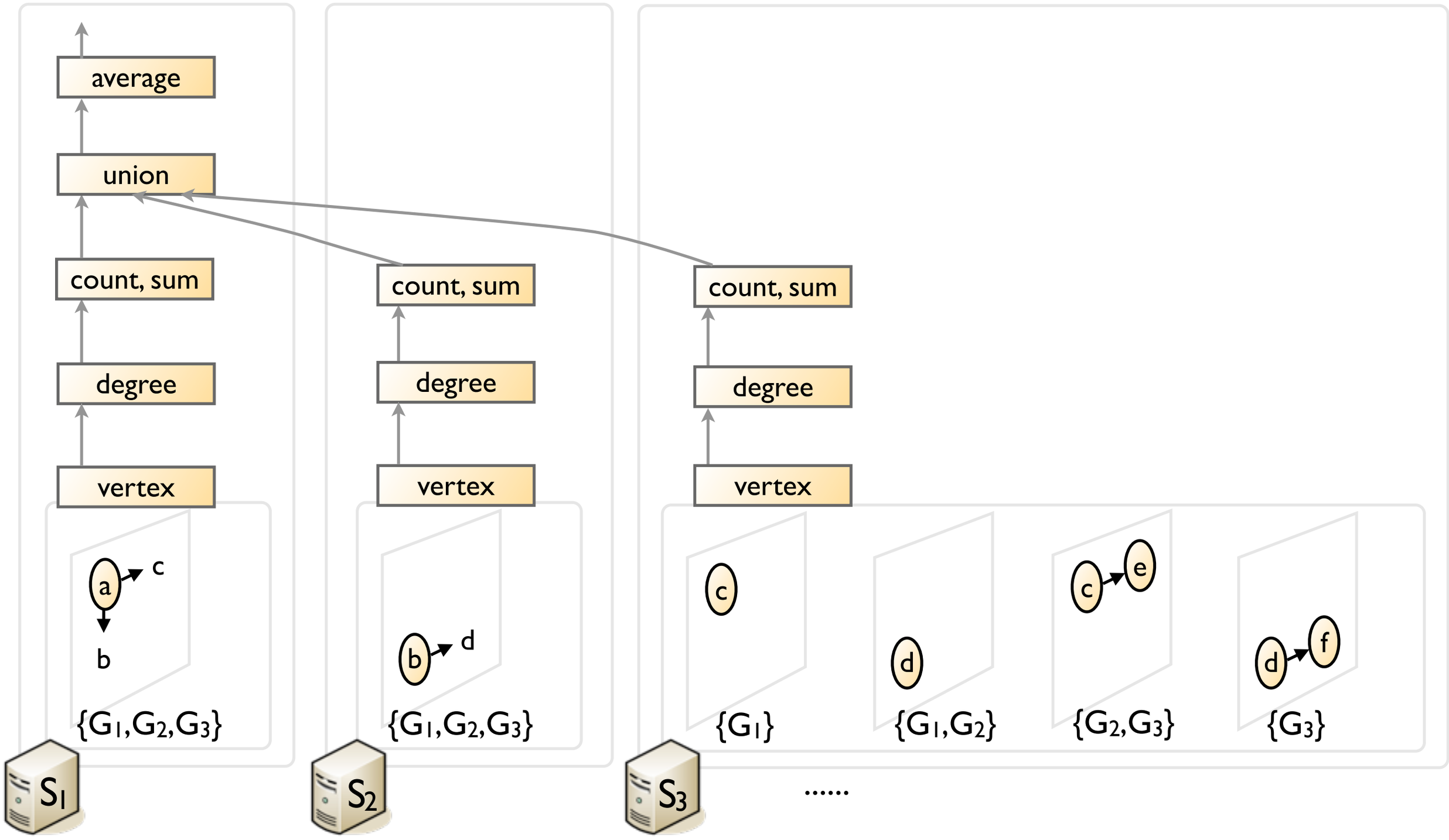
Example: average degree of each graph in $\{G_1, G_2\}$

Graph Query Execution



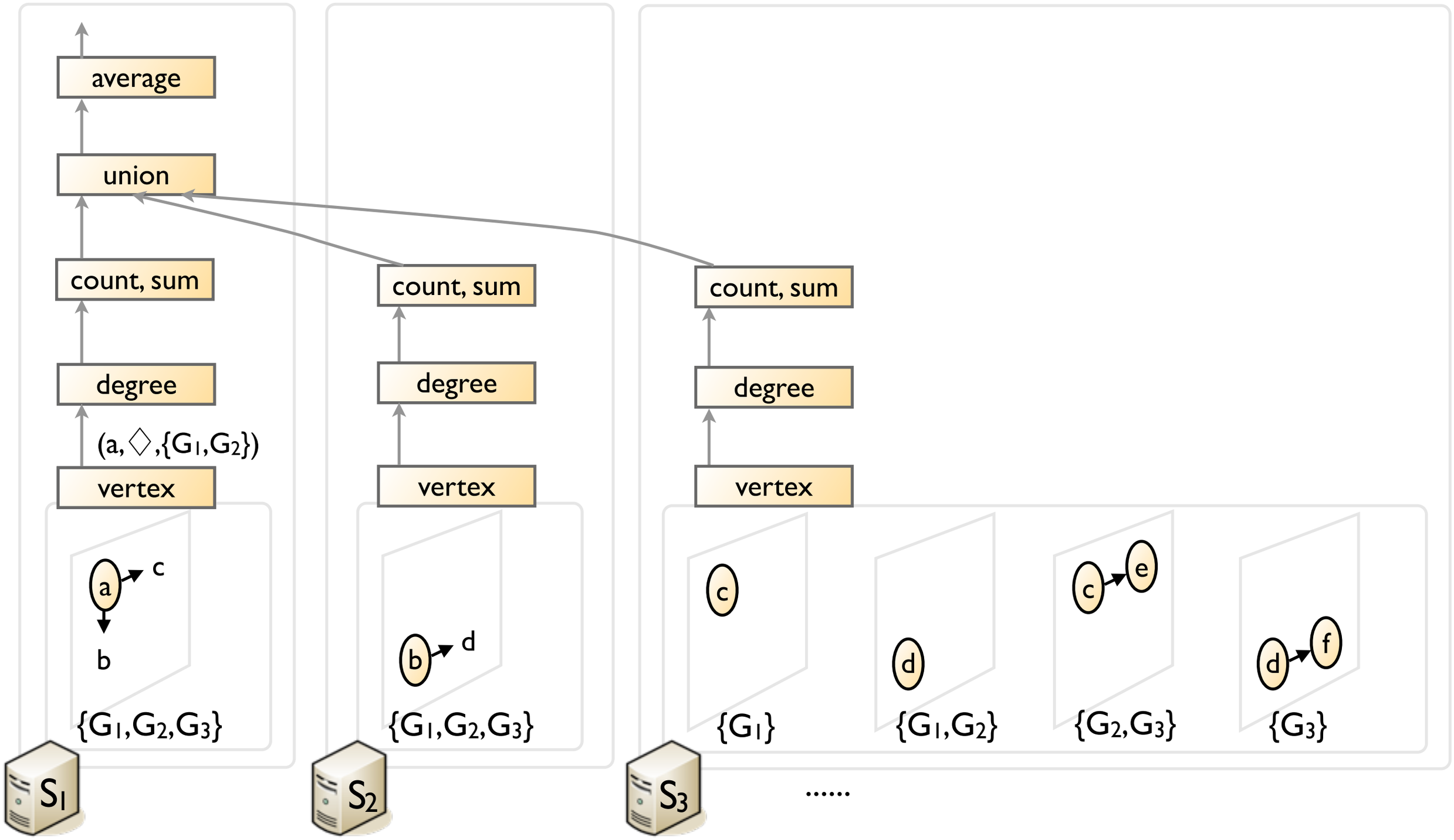
Example: average degree of each graph in $\{G_1, G_2\}$

Graph Query Execution



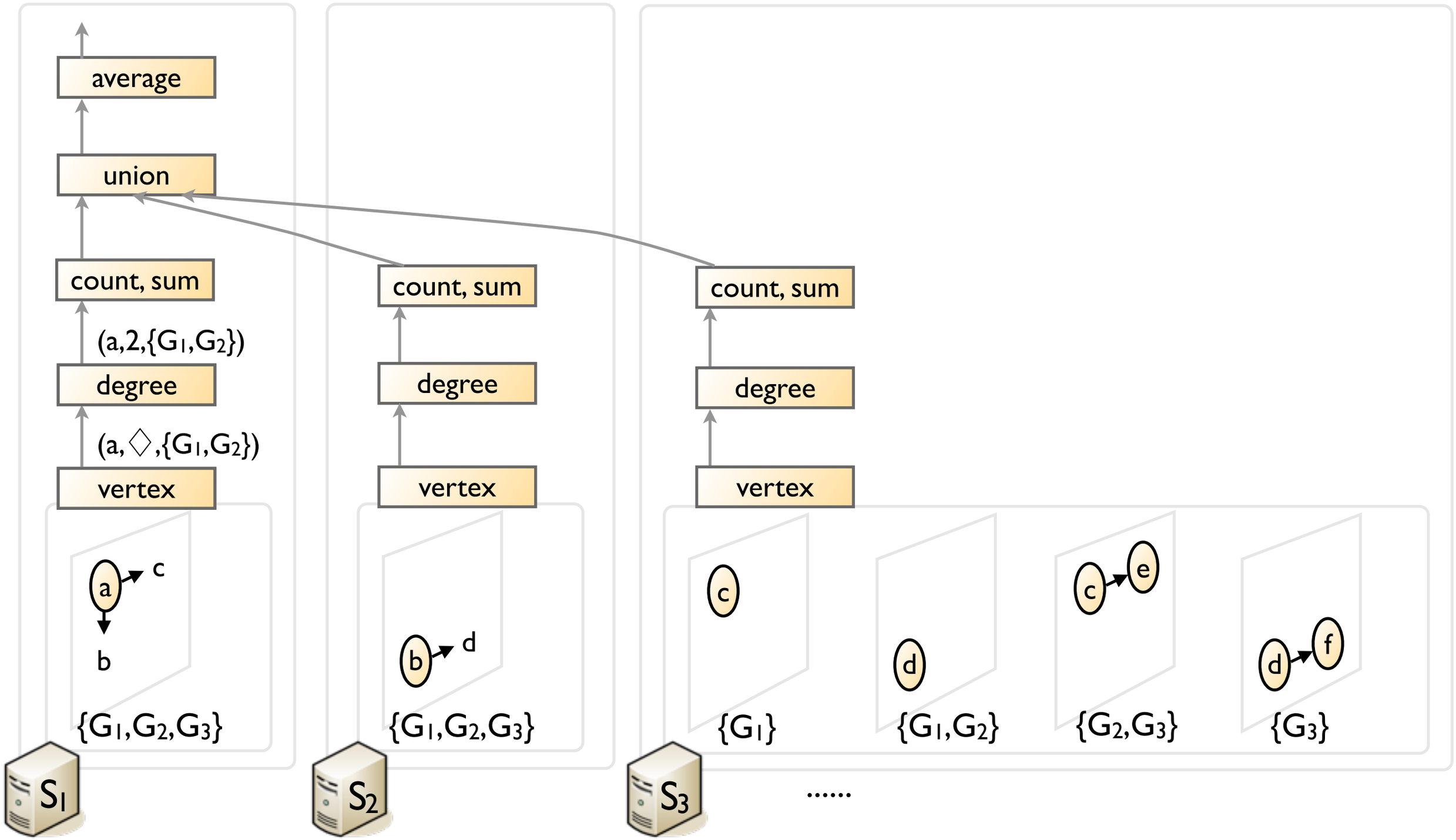
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



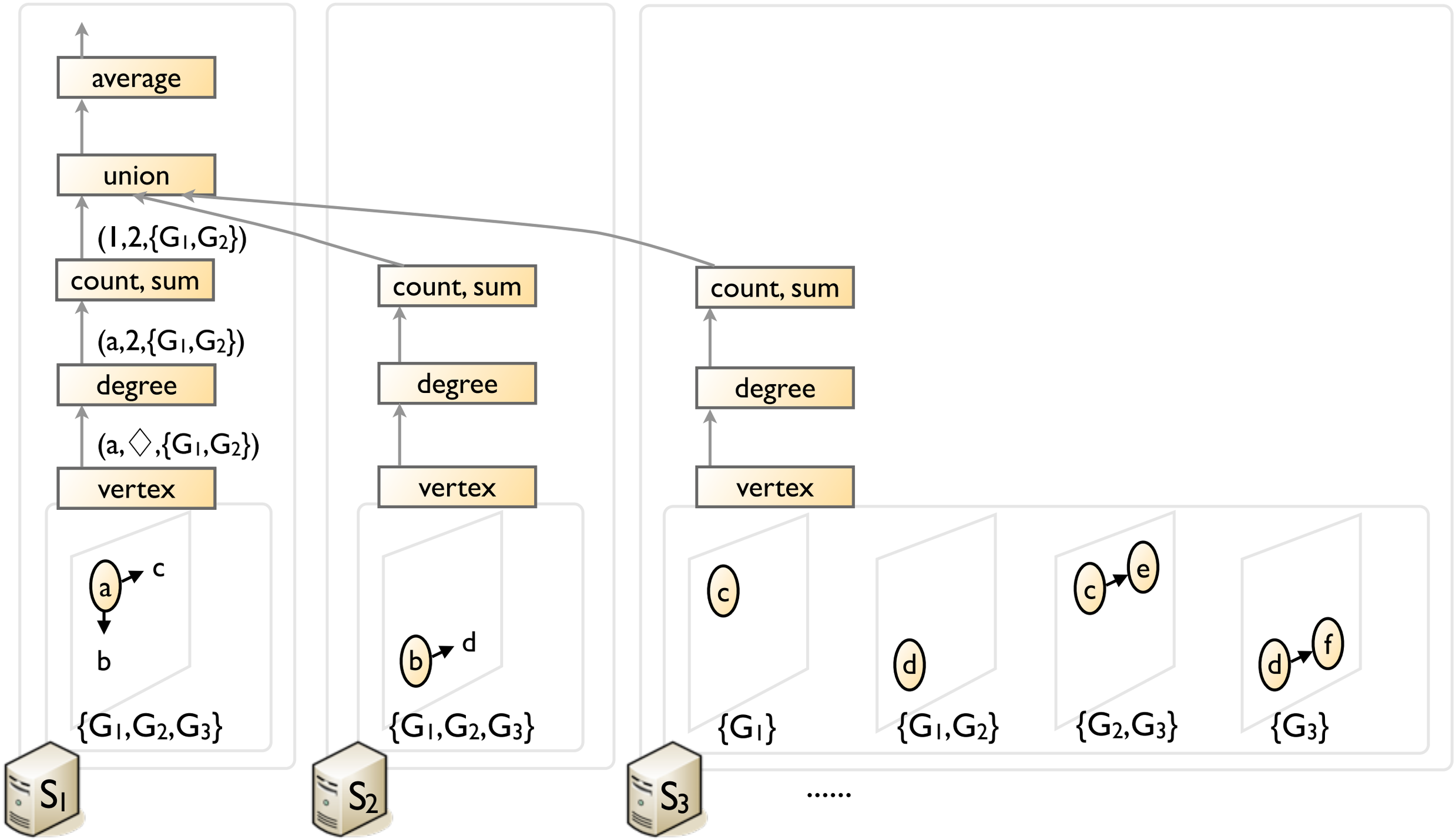
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



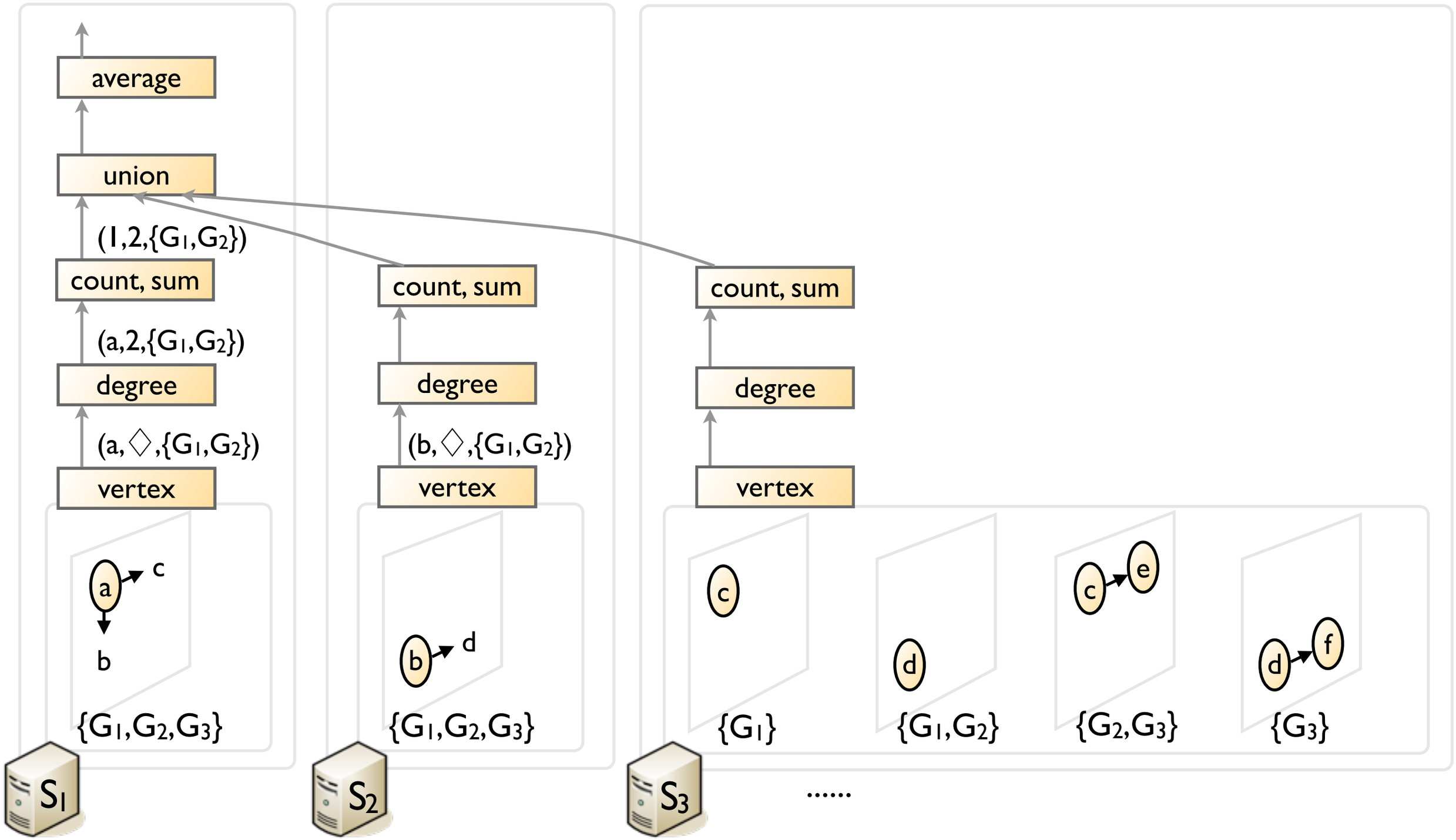
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



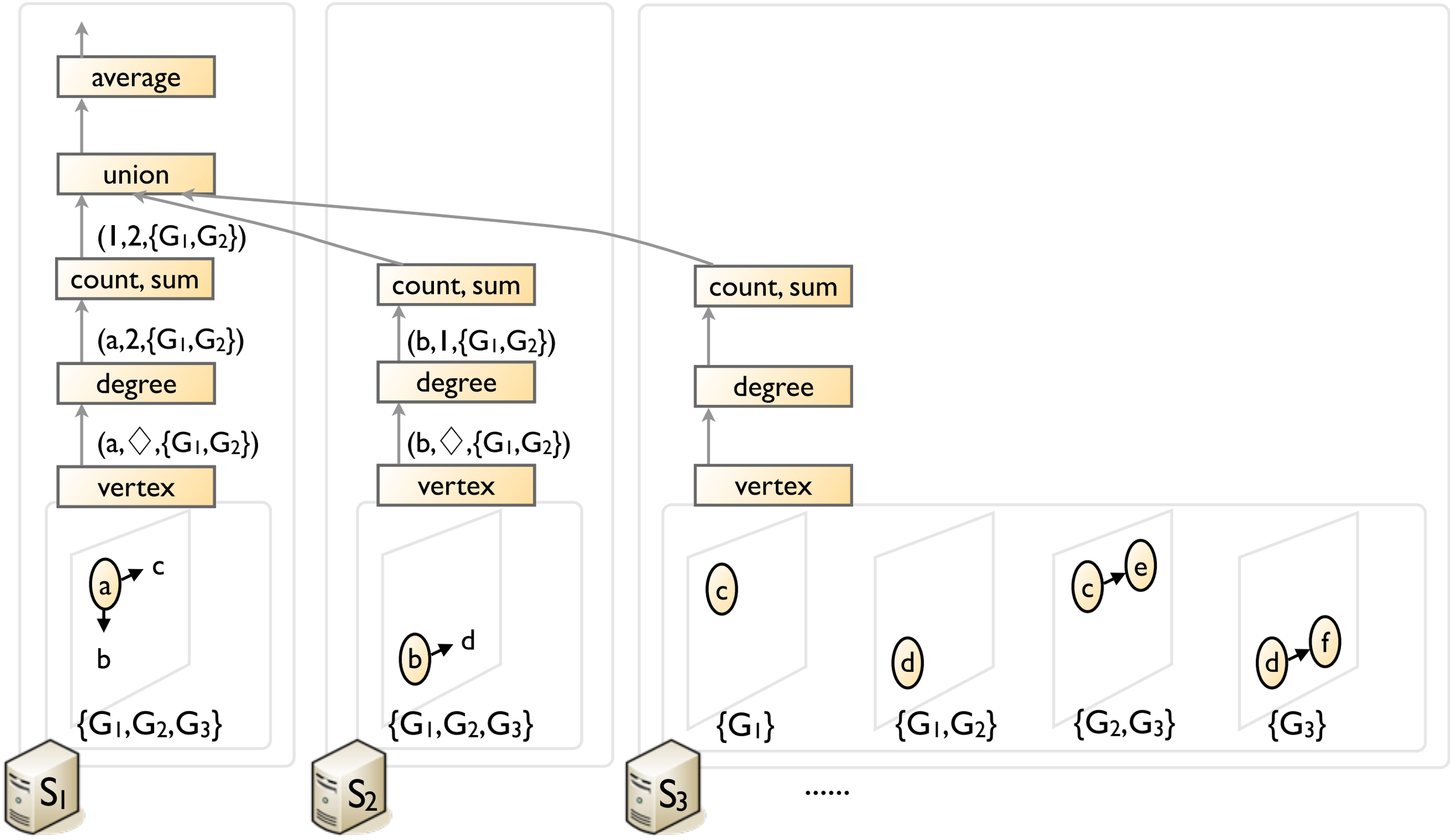
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



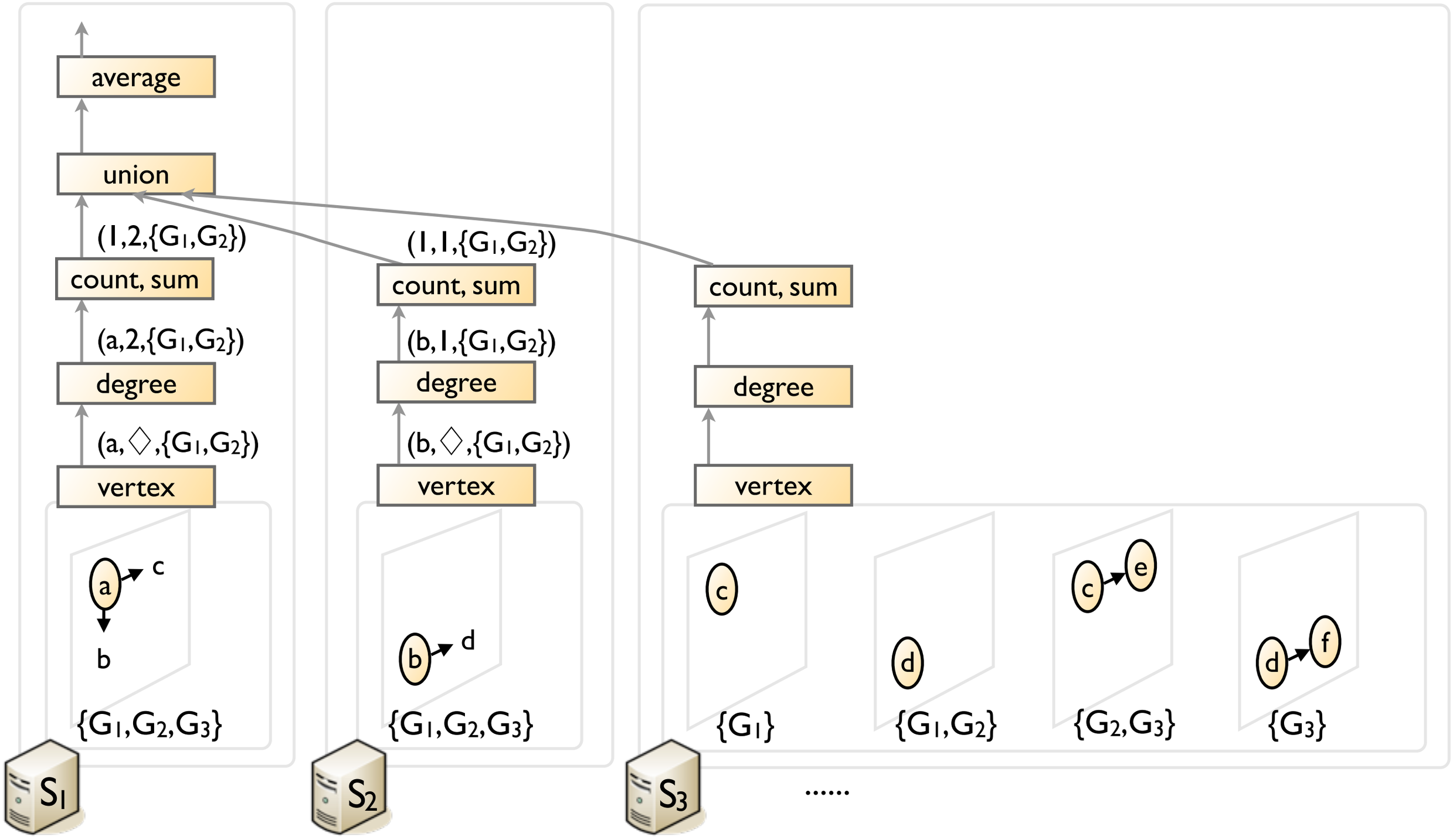
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



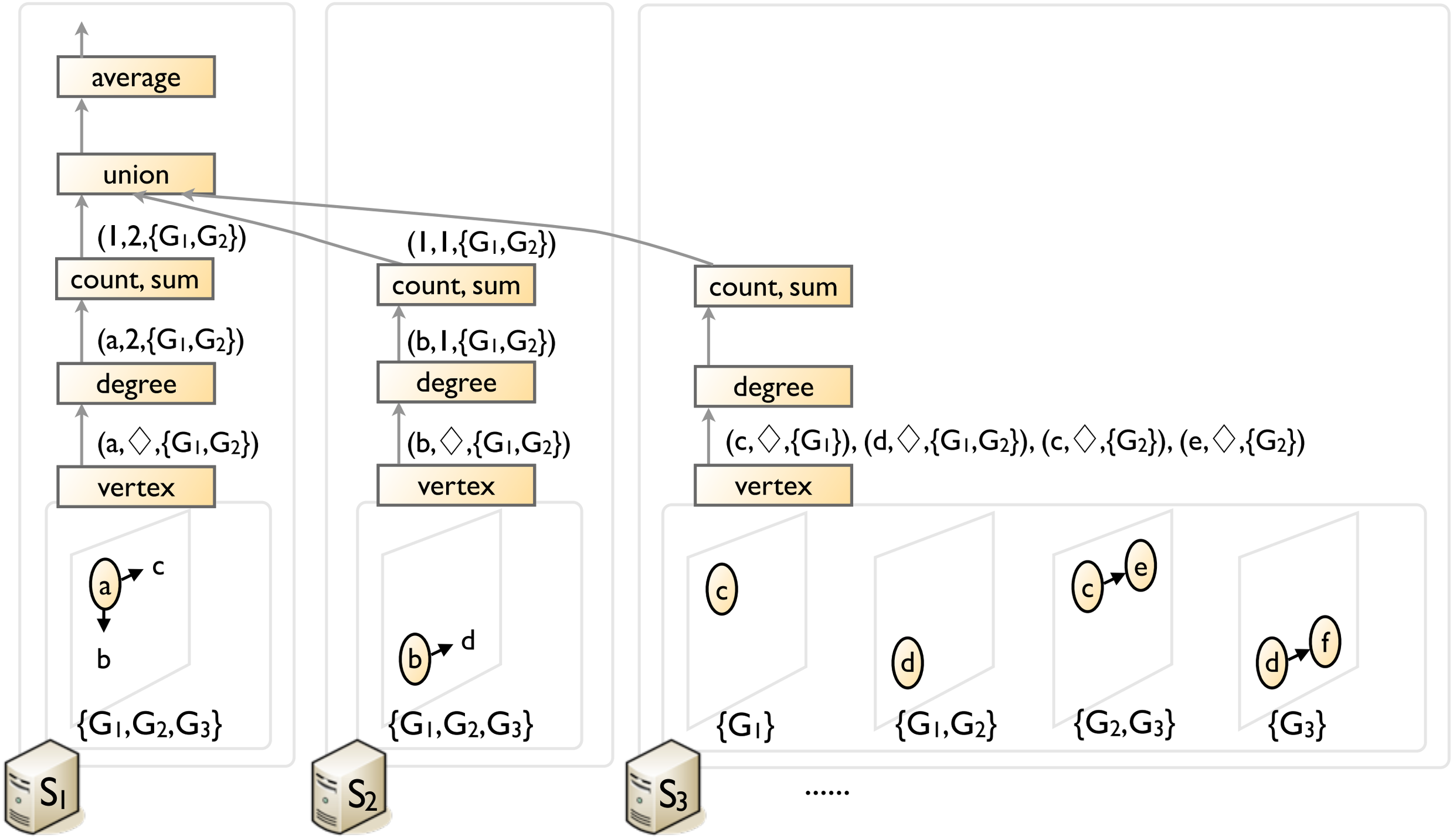
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



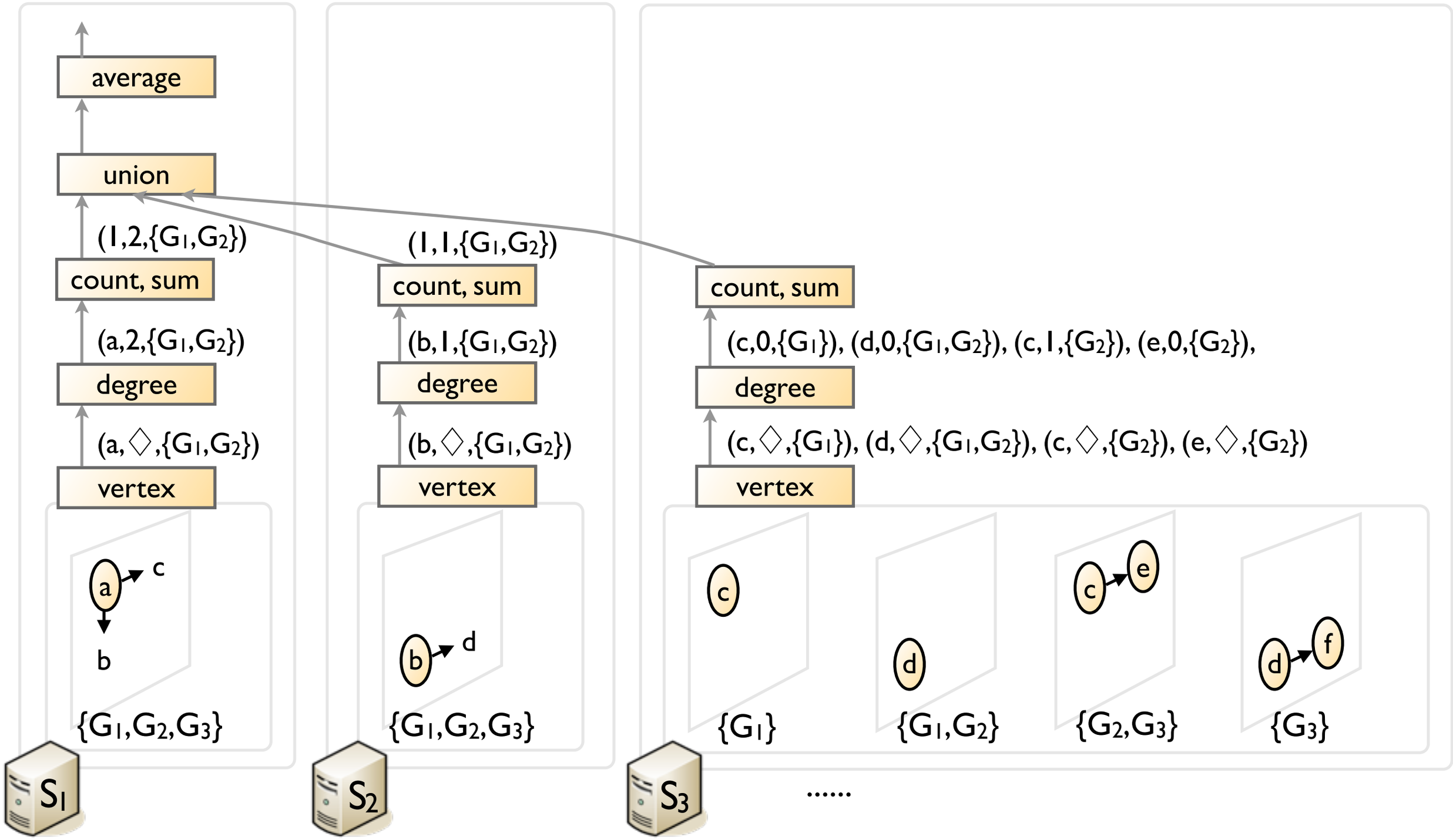
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



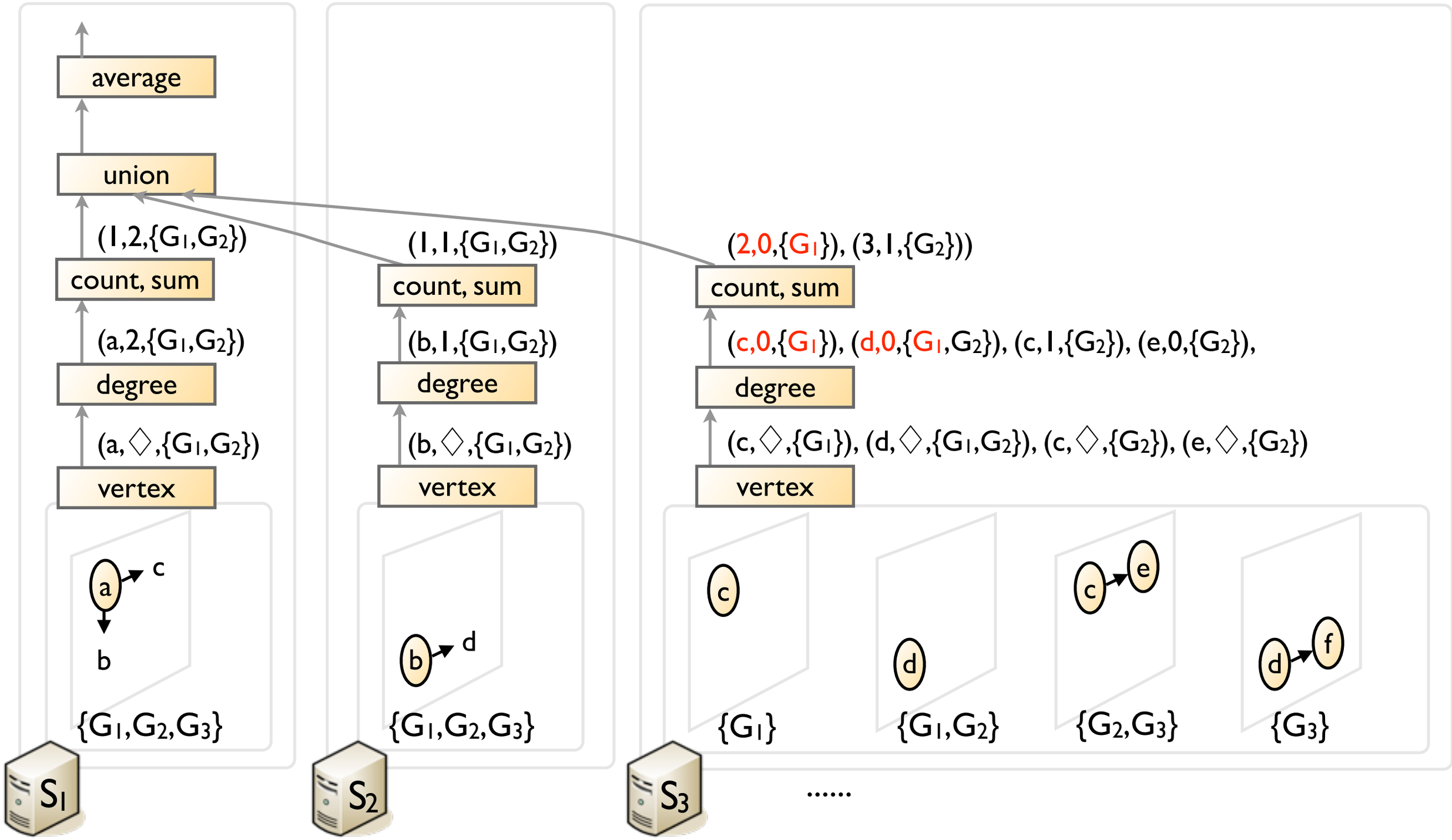
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



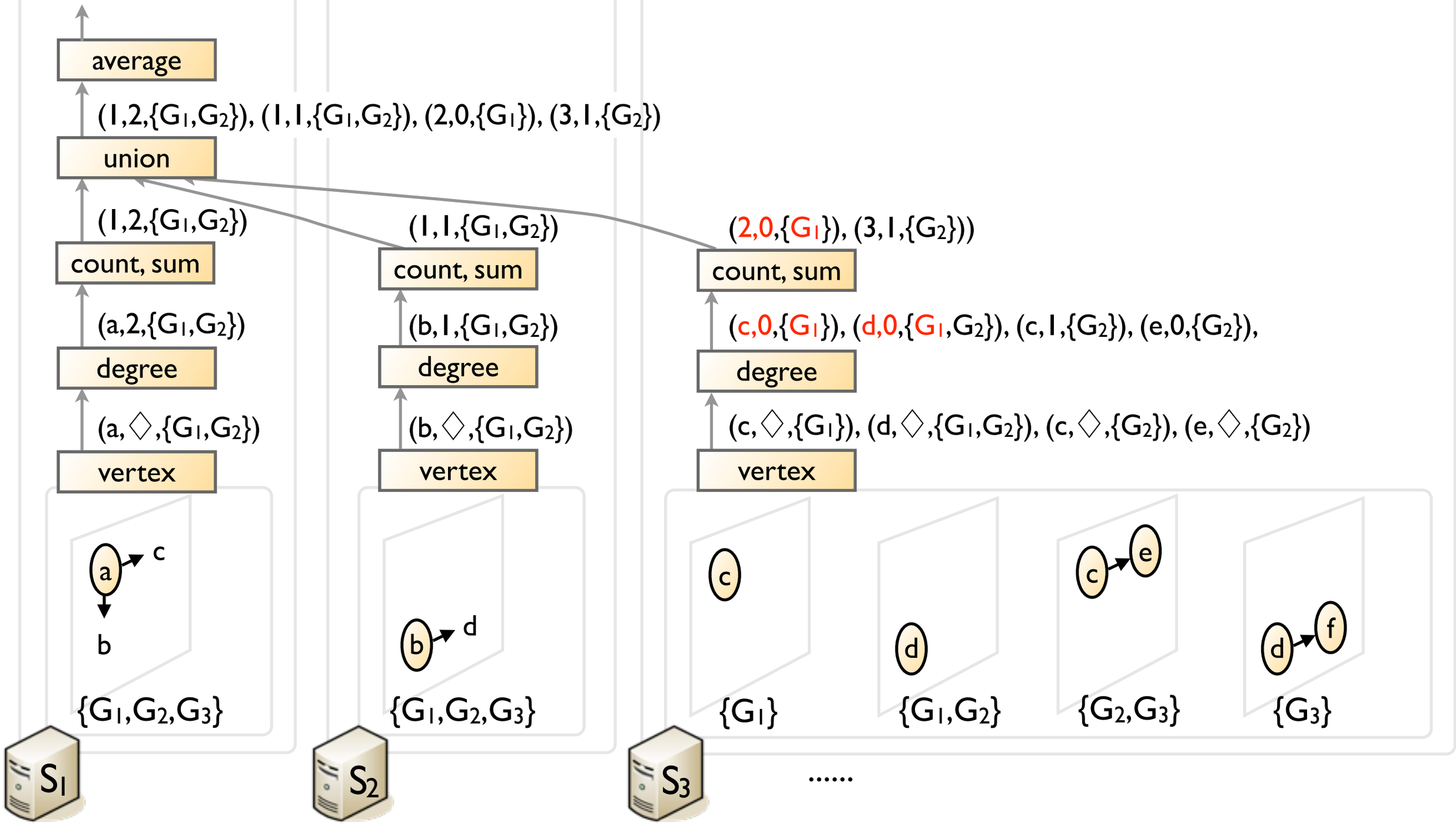
Example: average degree of each graph in $\{G_1, G_2\}$

Graph Query Execution



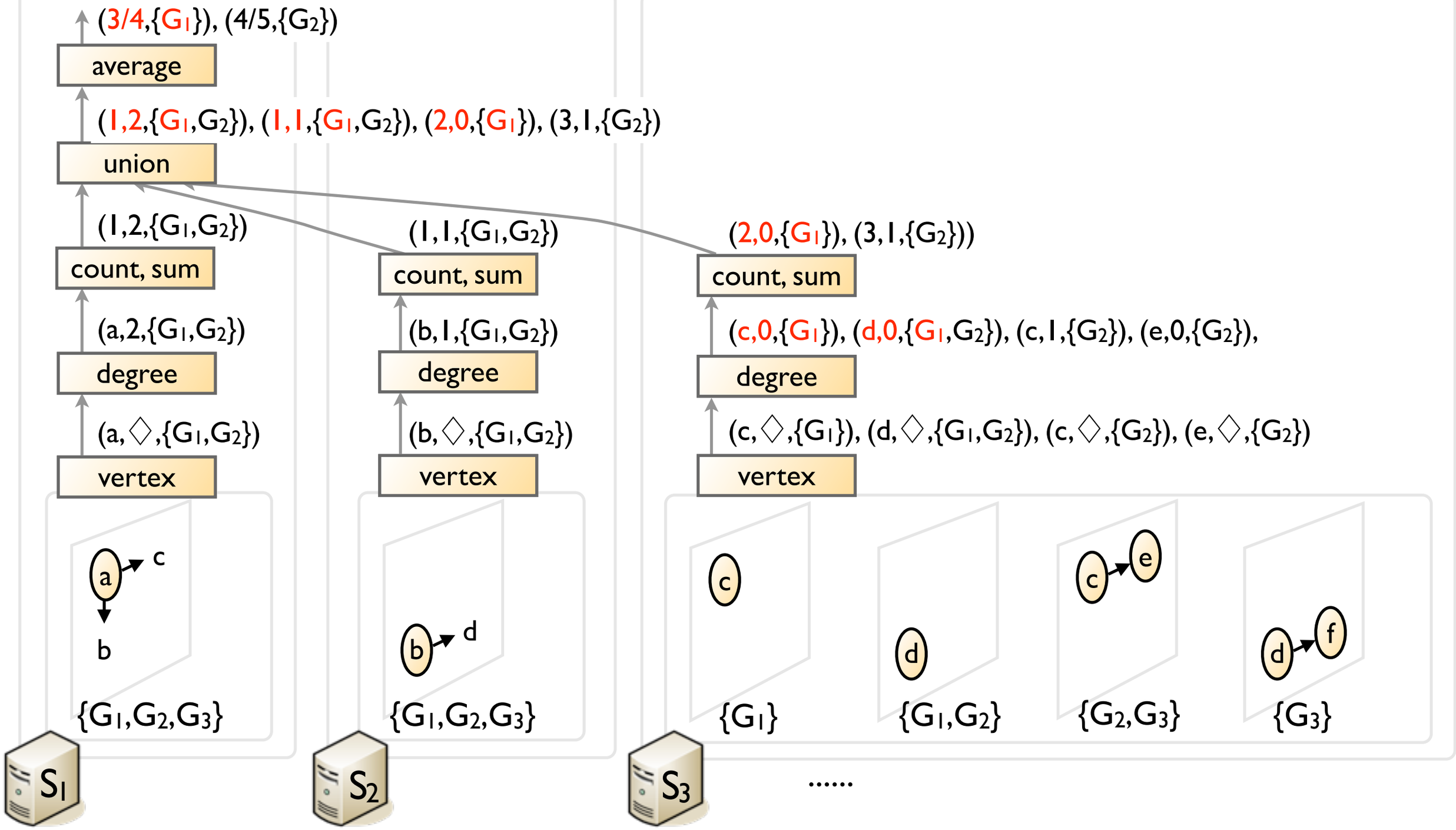
Example: average degree of each graph in $\{G_1, G_2\}$

Graph Query Execution



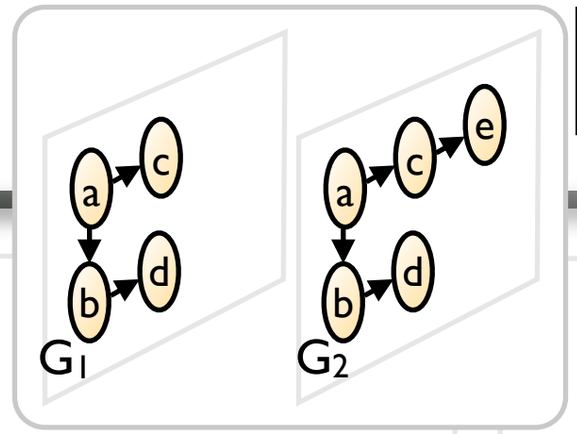
Example: average degree of each graph in {G_1,G_2}

Graph Query Execution



Example: average degree of each graph in {G_1,G_2}

Graph Execution



($3/4$,{G_1}), ($4/5$,{G_2})

average

($1,2$,{G_1,G_2}), ($1,1$,{G_1,G_2}), ($2,0$,{G_1}), ($3,1$,{G_2})

union

($1,2$,{G_1,G_2})

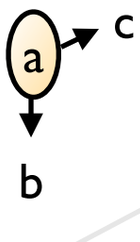
count, sum

($a,2$,{G_1,G_2})

degree

(a,◇,{G_1,G_2})

vertex



{G_1,G_2,G_3}

S_1

($1,1$,{G_1,G_2})

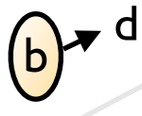
count, sum

($b,1$,{G_1,G_2})

degree

(b,◇,{G_1,G_2})

vertex



{G_1,G_2,G_3}

S_2

($2,0$,{G_1}), ($3,1$,{G_2}))

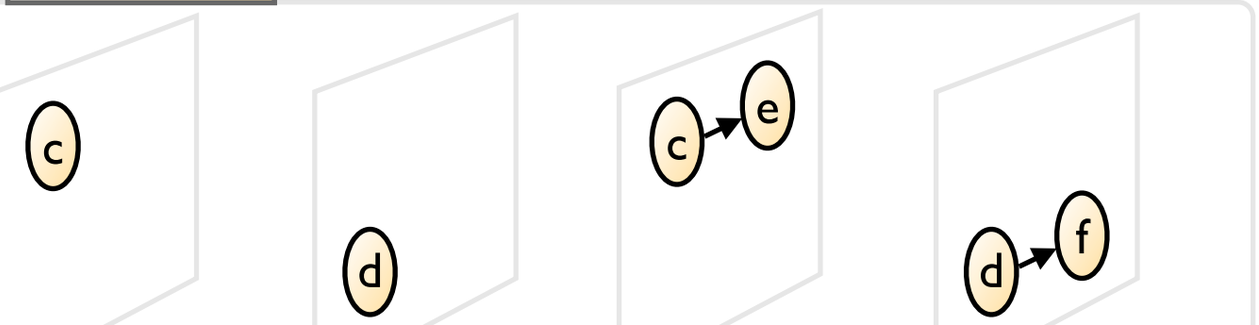
count, sum

($c,0$,{G_1}), ($d,0$,{G_1,G_2}), ($c,1$,{G_2}), ($e,0$,{G_2}),

degree

(c,◇,{G_1}), (d,◇,{G_1,G_2}), (c,◇,{G_2}), (e,◇,{G_2})

vertex



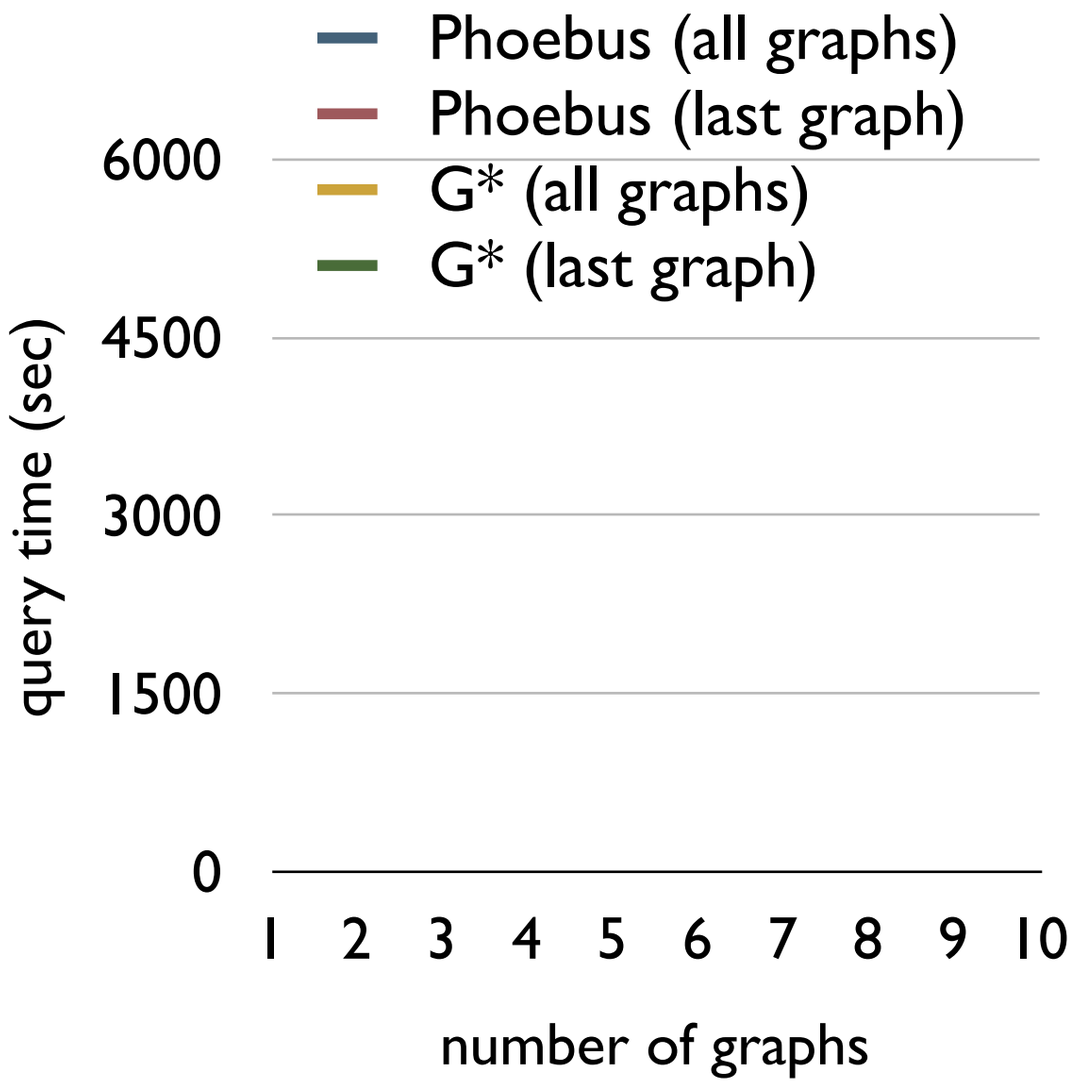
{G_1} {G_1,G_2} {G_2,G_3} {G_3}

S_3

.....

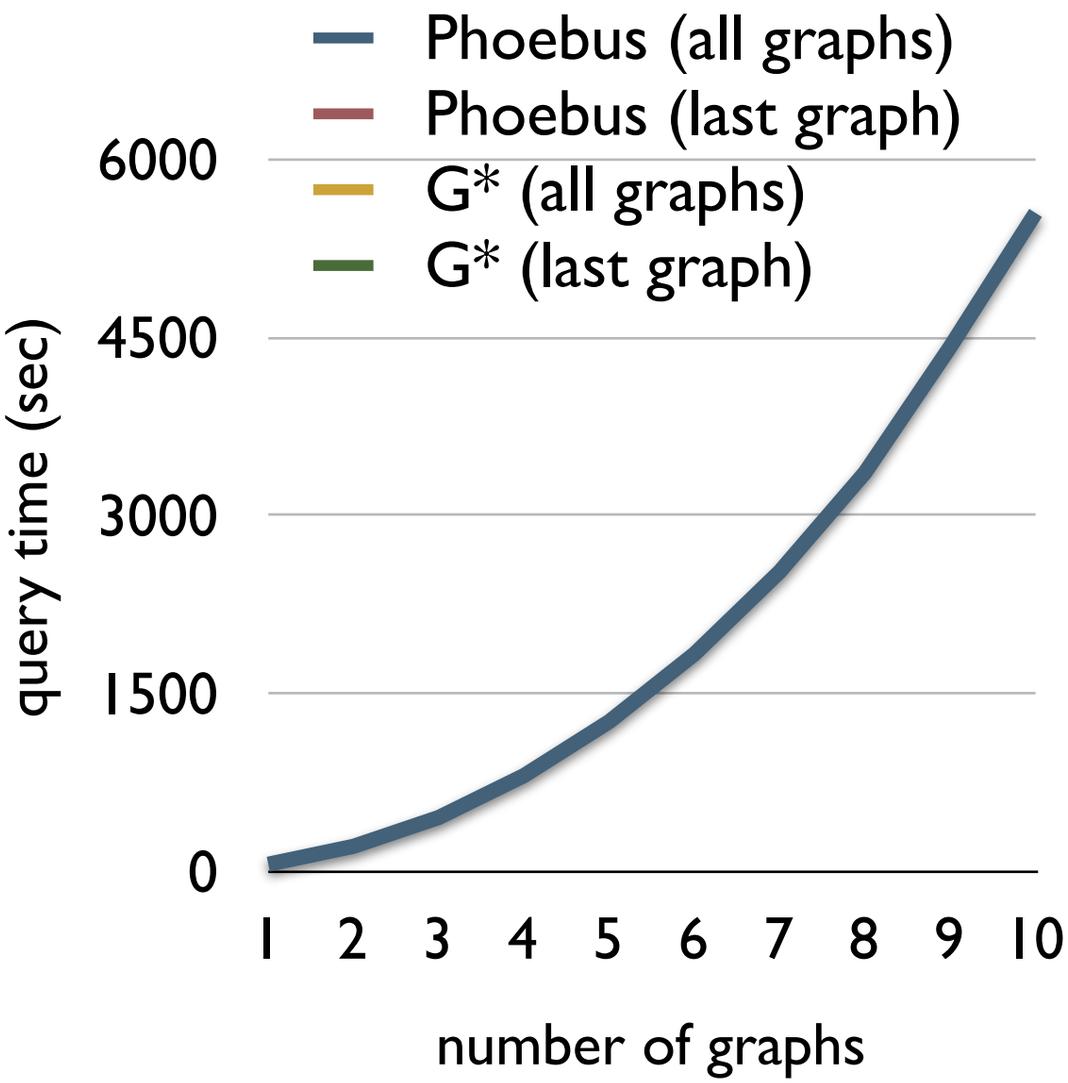
G* vs Phoebus (an open-source version of Pregel)

- Data
  - 10 complete binary trees
  - i'th tree = (i-1)'th tree
    +
    new 25K vertices (and edges)
- Query
  - single-source shortest paths



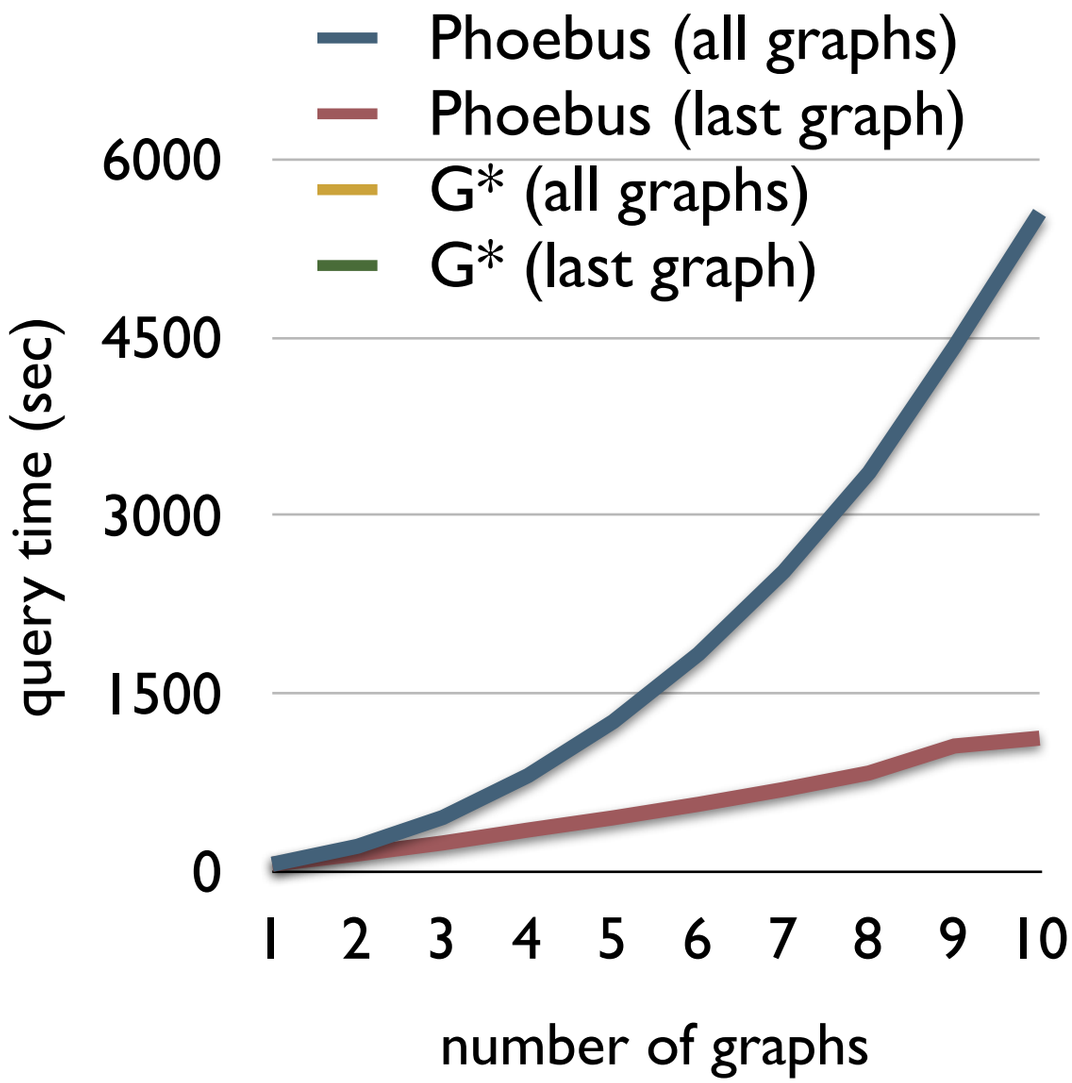
G* vs Phoebus (an open-source version of Pregel)

- Data
  - 10 complete binary trees
  - i'th tree = (i-1)'th tree
    +
    new 25K vertices (and edges)
- Query
  - single-source shortest paths



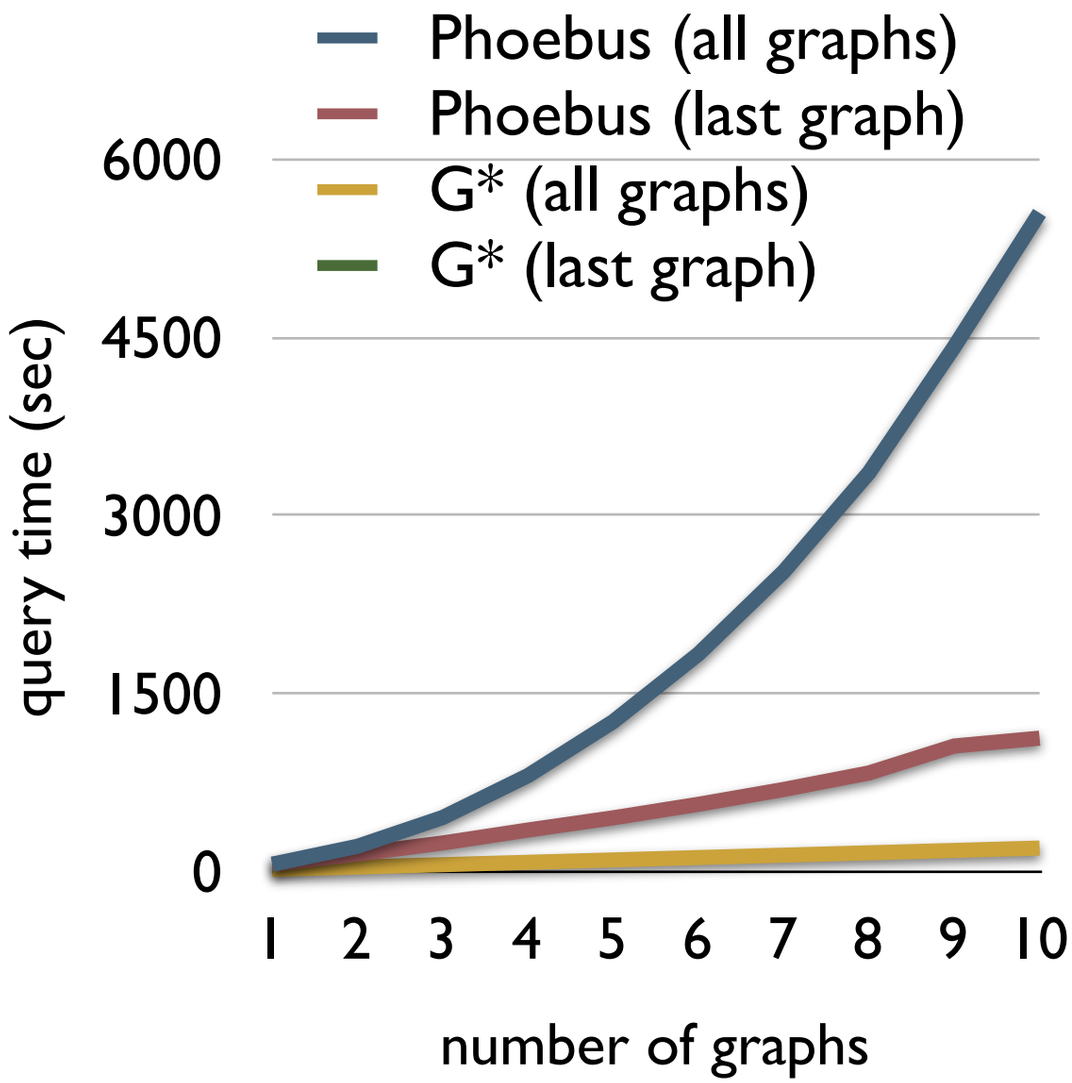
G* vs Phoebus (an open-source version of Pregel)

- Data
  - 10 complete binary trees
  - i'th tree = (i-1)'th tree
    +
    new 25K vertices (and edges)
- Query
  - single-source shortest paths



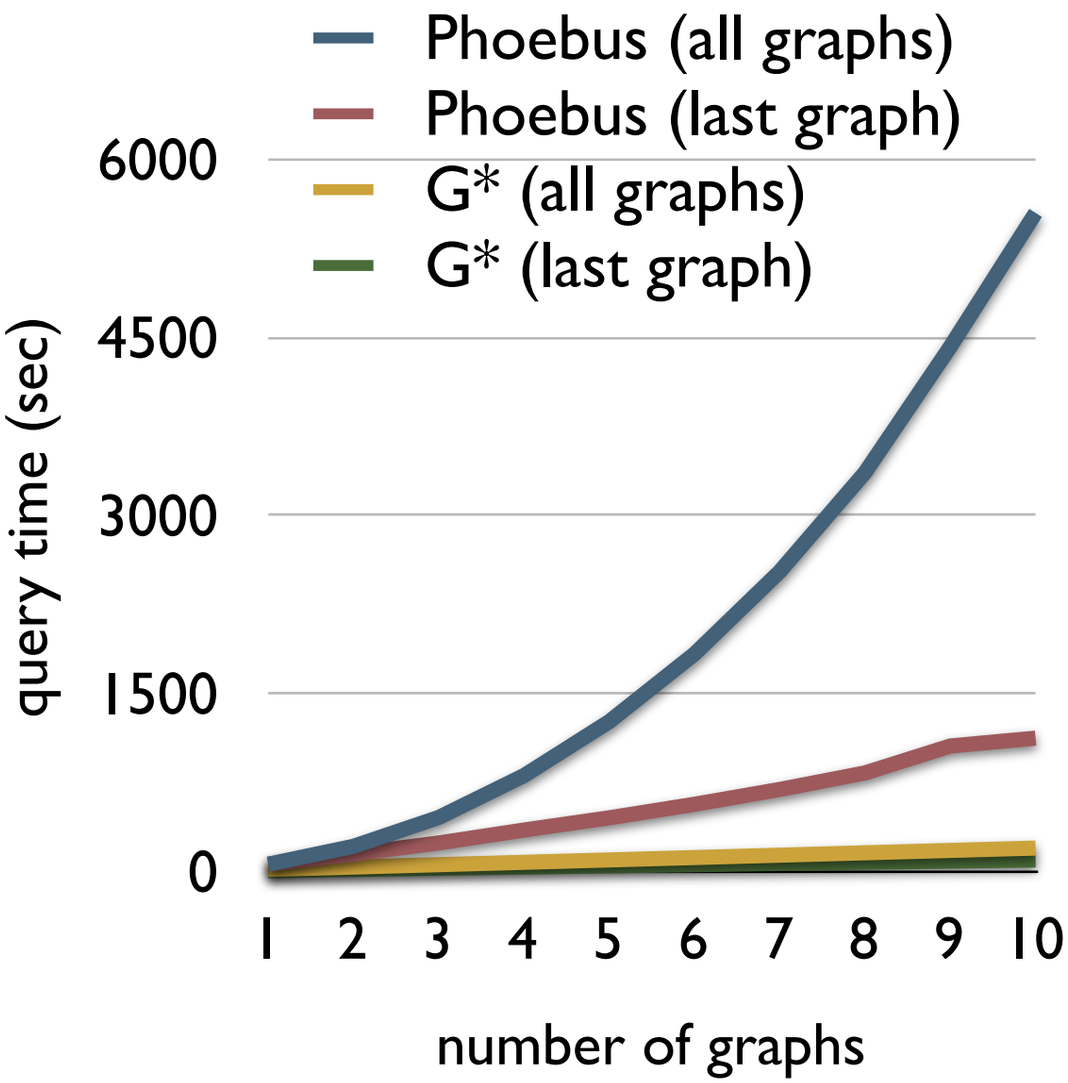
G* vs Phoebus (an open-source version of Pregel)

- Data
  - 10 complete binary trees
  - i'th tree = (i-1)'th tree
    +
    new 25K vertices (and edges)
- Query
  - single-source shortest paths



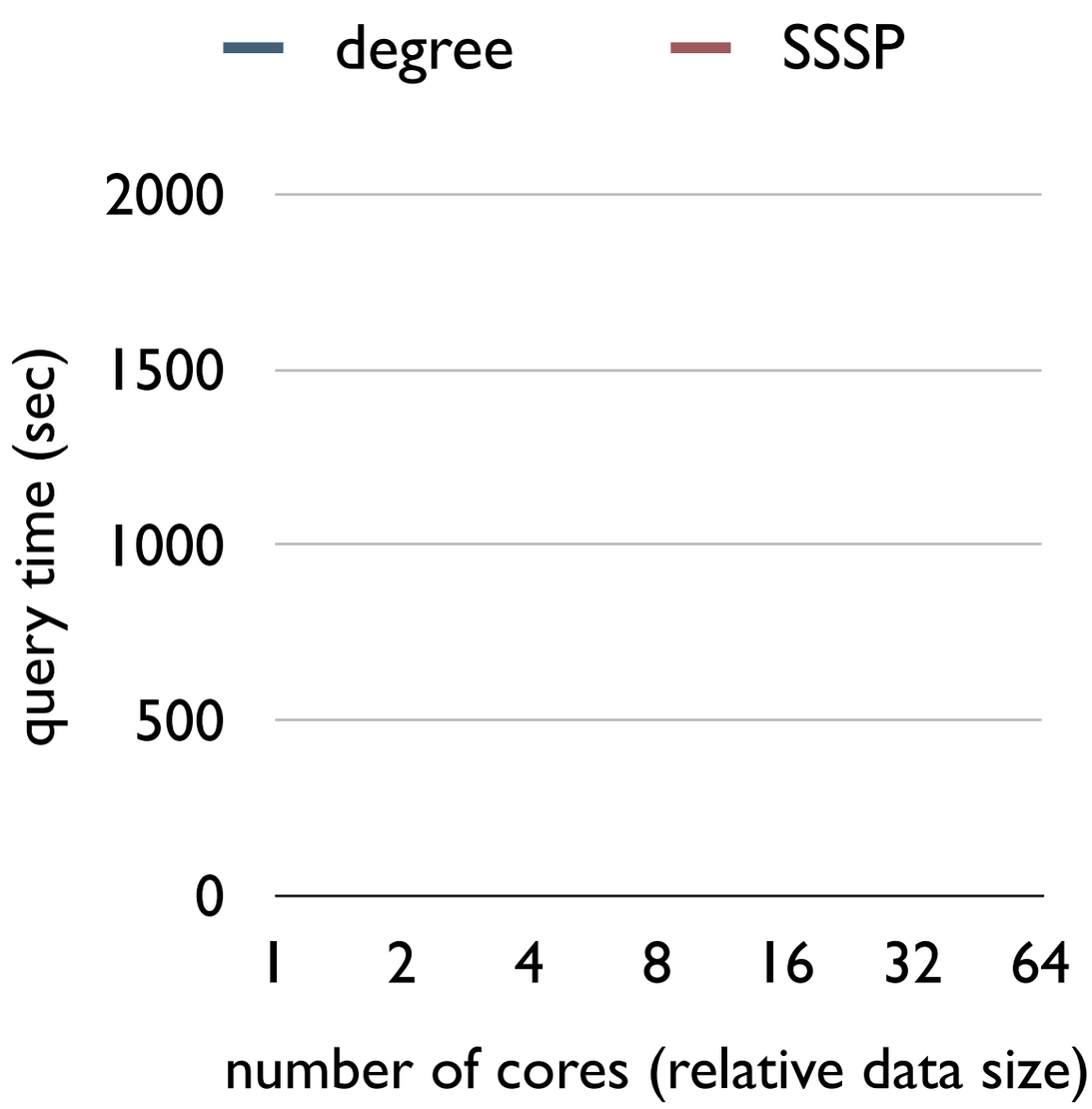
G* vs Phoebus (an open-source version of Pregel)

- Data
  - 10 complete binary trees
  - i'th tree = (i-1)'th tree
    +
    new 25K vertices (and edges)
- Query
  - single-source shortest paths



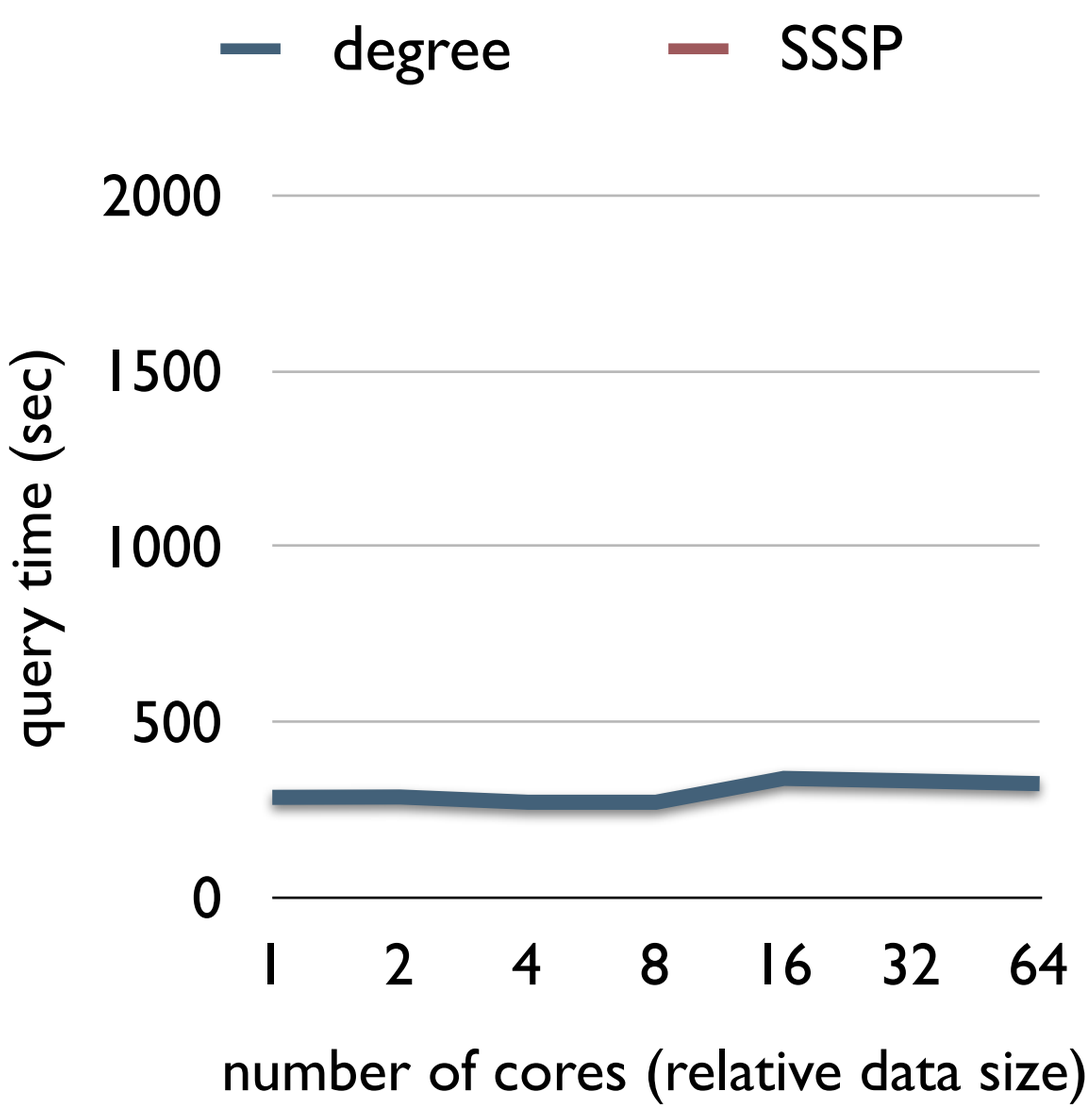
Scale-up Test

- Platform
  - 64 core server cluster
- Data
  - 1 complete binary tree
  - 1 billion vertices (edges)
  - 64 million vertices / server
- Queries
  - vertex degree
  - single-source shortest paths



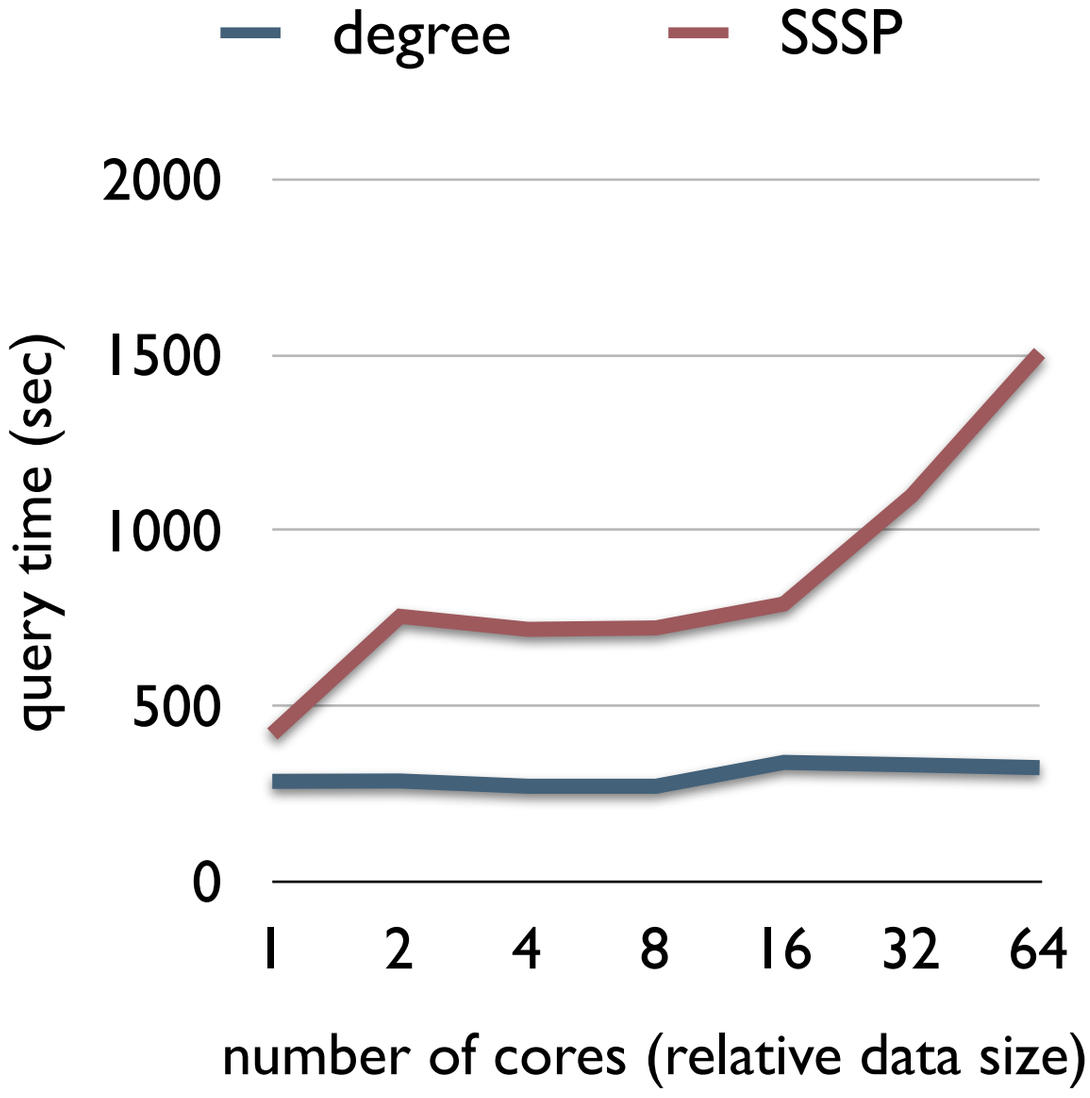
Scale-up Test

- Platform
  - 64 core server cluster
- Data
  - 1 complete binary tree
  - 1 billion vertices (edges)
  - 64 million vertices / server
- Queries
  - vertex degree
  - single-source shortest paths



Scale-up Test

- Platform
  - 64 core server cluster
- Data
  - 1 complete binary tree
  - 1 billion vertices (edges)
  - 64 million vertices / server
- Queries
  - vertex degree
  - single-source shortest paths



Conclusion

- Convenient and efficient queries on graphs
  - distributed, deduplicate storage of graphs
  - parallel query processing that share computations across graphs
- Supported by NSF CAREER award IIS-1149372
- 30K lines of Java code
- 1st code release in August 2012
- Future work
  - disk optimization / graph distribution / query optimization / high availability

Thank you!