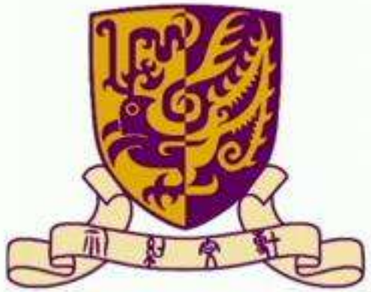
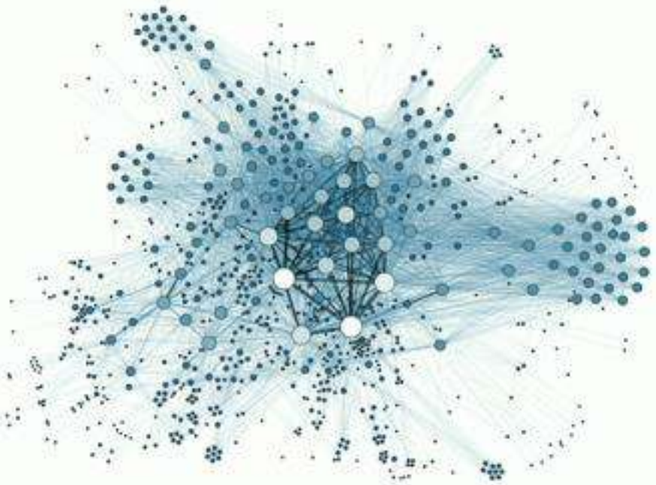
Optimizing Declarative Graph Queries at Large Scale

Qizhen Zhang, Akash Acharya, Hongzhi Chen⁺, Simran Arora, Ang Chen*, Vincent Liu, Boon Thau Loo

*University of Pennsylvania, ⁺The Chinese University of Hong Kong, *Rice University*



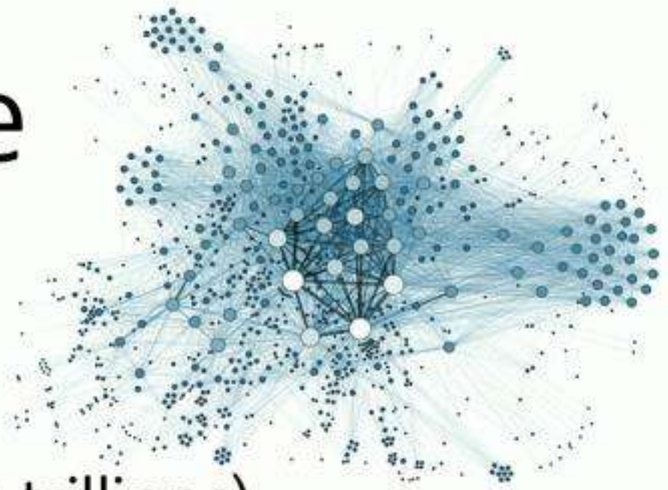
Graphs Are Important



Graph processing is everywhere

- Web graph, social networks, road networks…

Graphs Are Important & Large



Graph processing is everywhere

- Web graph, social networks, road networks…
- They are increasingly large (scaling from billions to trillions).

The largest workloads are deployed at massive scales

- Clouds
- Data centers

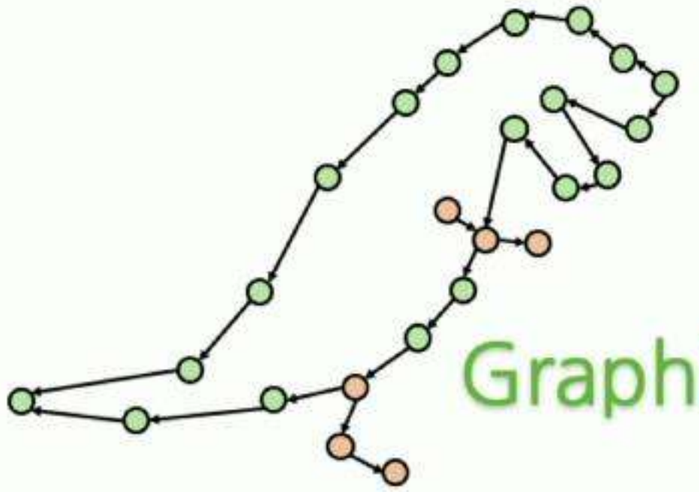
Large deployments require:

- Robustness
- Ease of programming
- Performance
- *Achieving three is difficult*

Challenges at Large Scales

- **The impact of real-world graph characteristics**
  - Power-law distribution: the optimal query execution may differ based on which part of the graph is being processed.
  - Dense connectivity: significant duplication.
- **The impact of data center characteristics**
  - Oversubscription
  - Link failures
  - Background traffic, etc.

GraphRex: Graph Recursive Execution

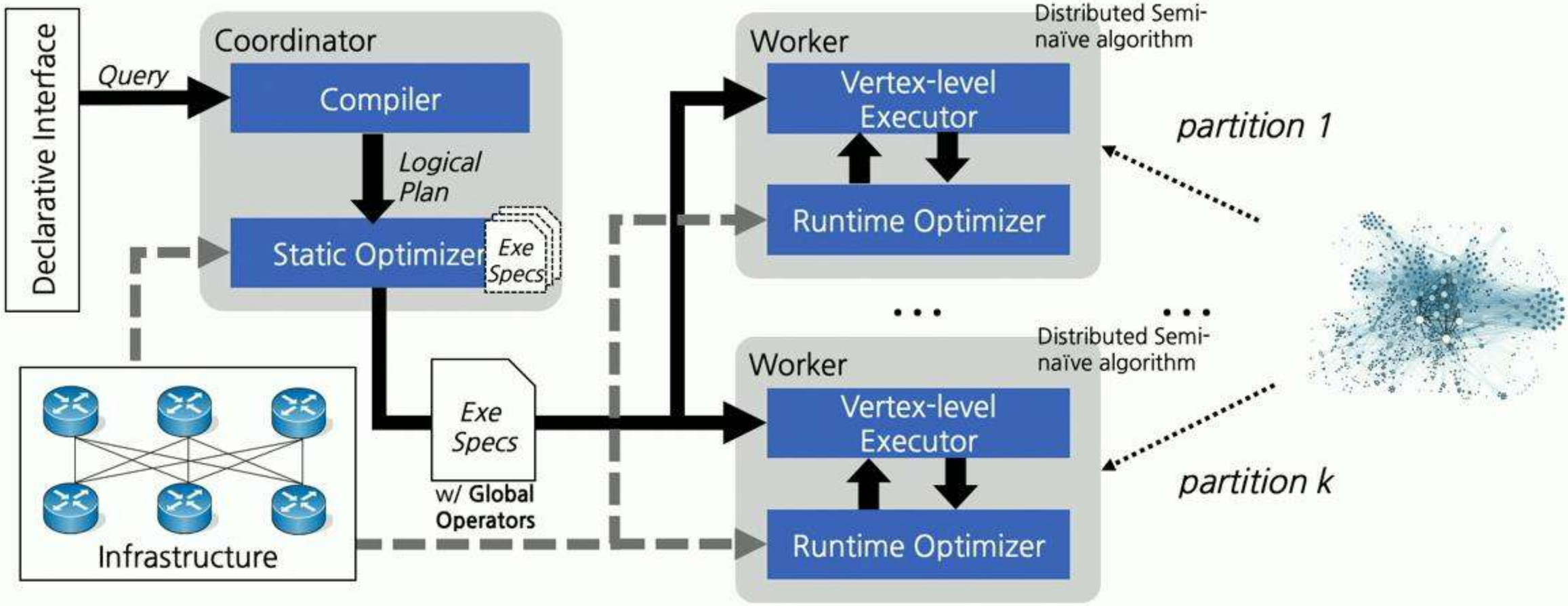


Datalog-like interface — (Ease of programming)

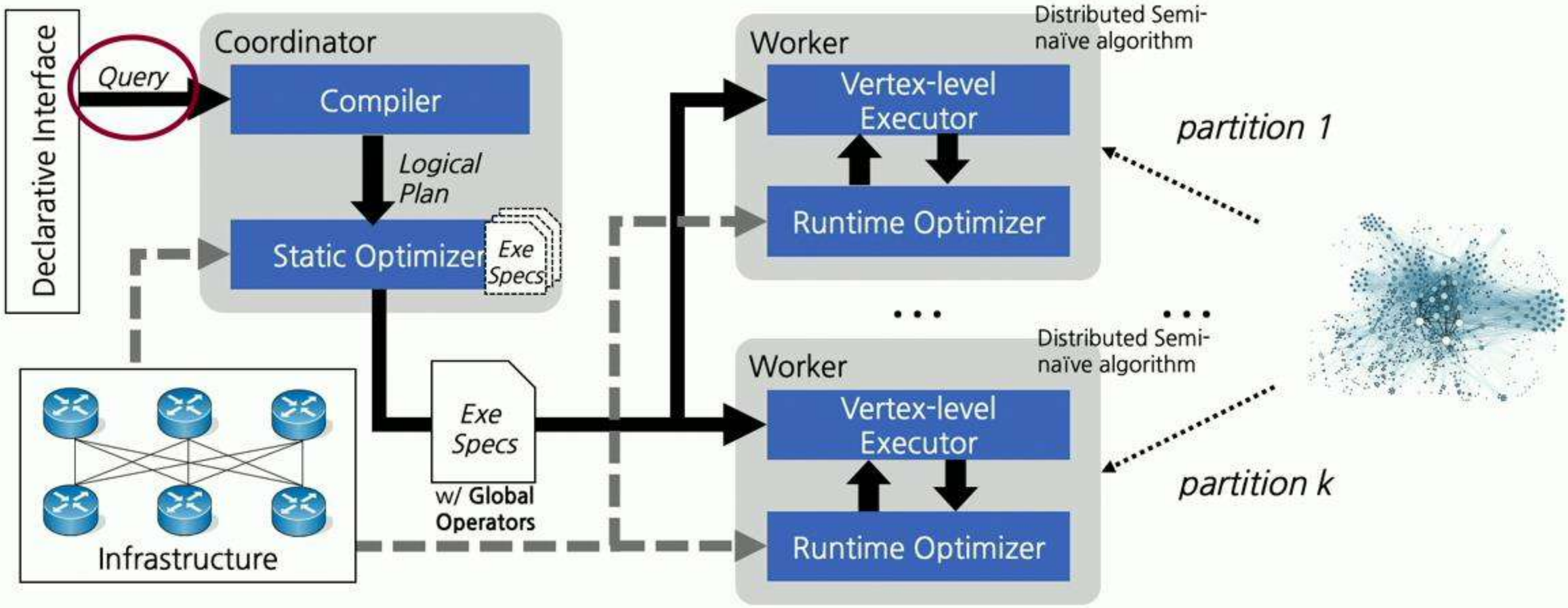
The knowledge of graphs — (Performance)

The knowledge of data center networks — (Performance, Robustness)

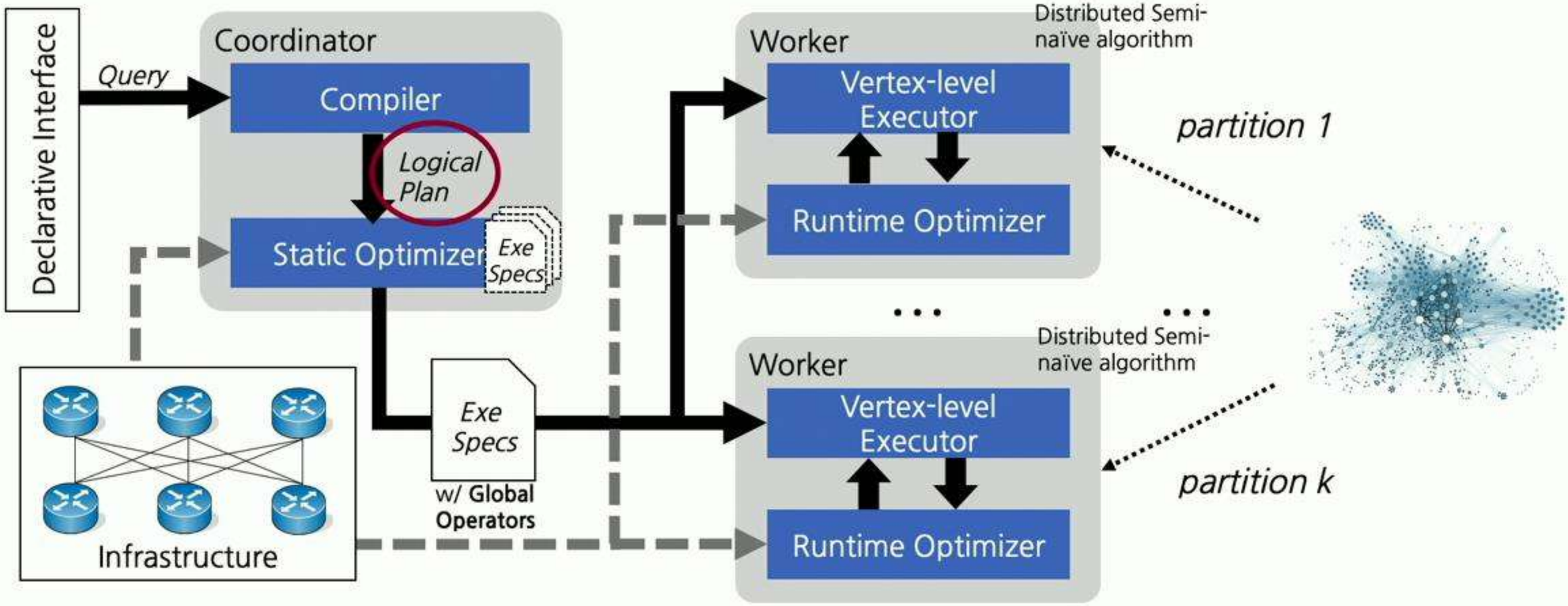
GraphRex Overview



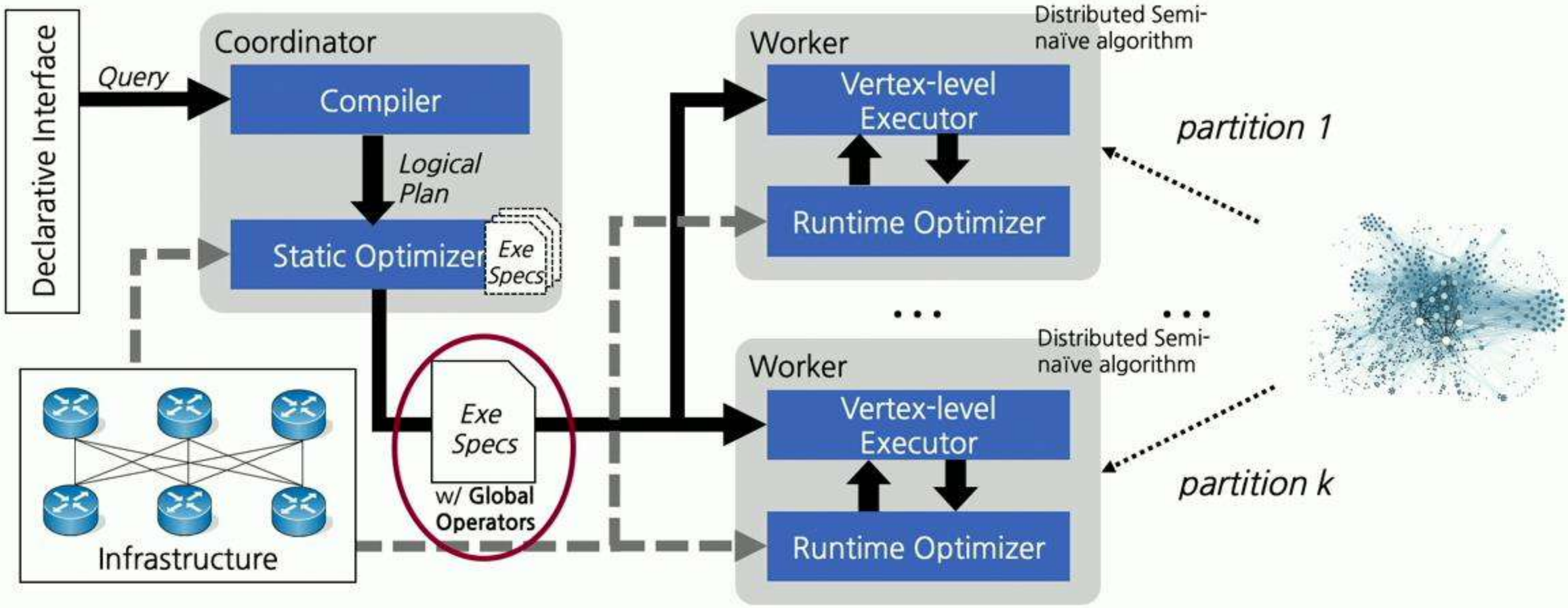
GraphRex Overview



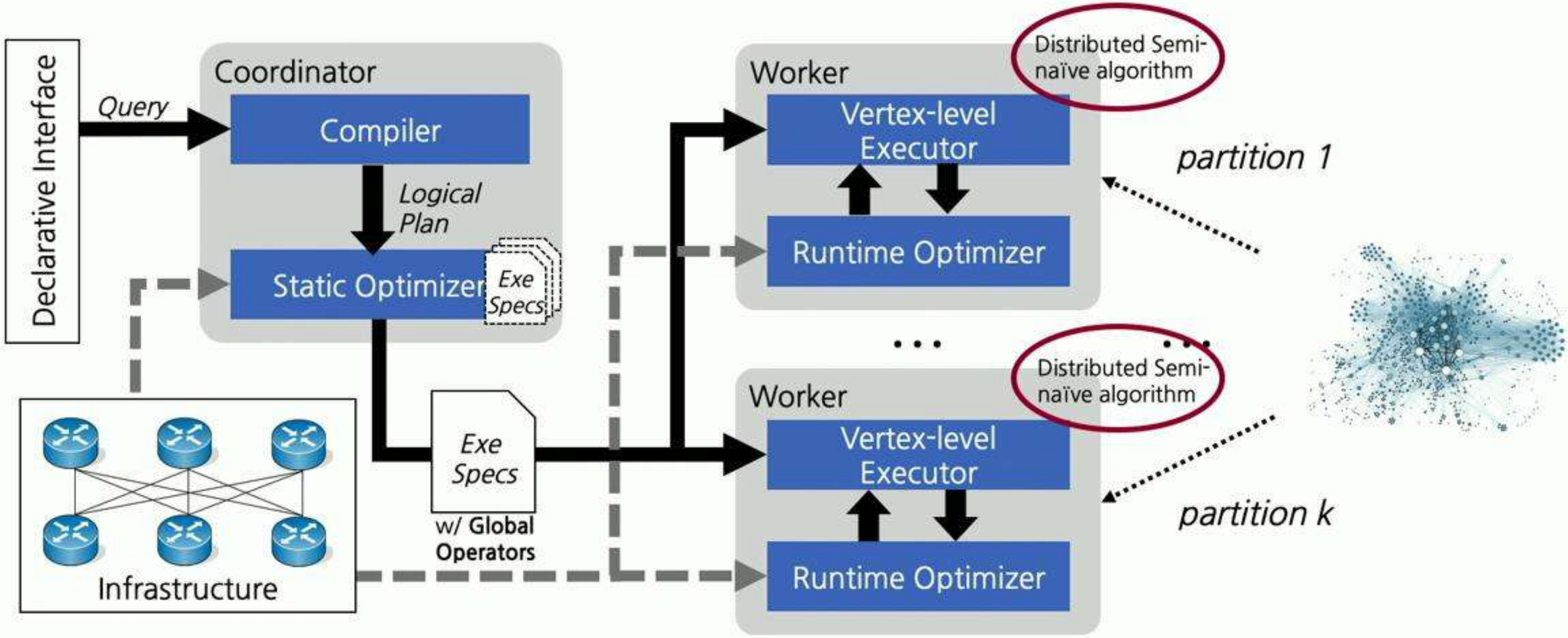
GraphRex Overview



GraphRex Overview



GraphRex Overview



Outline

Motivation & Proposal

Data Center Background

Data Center-Centric Optimization

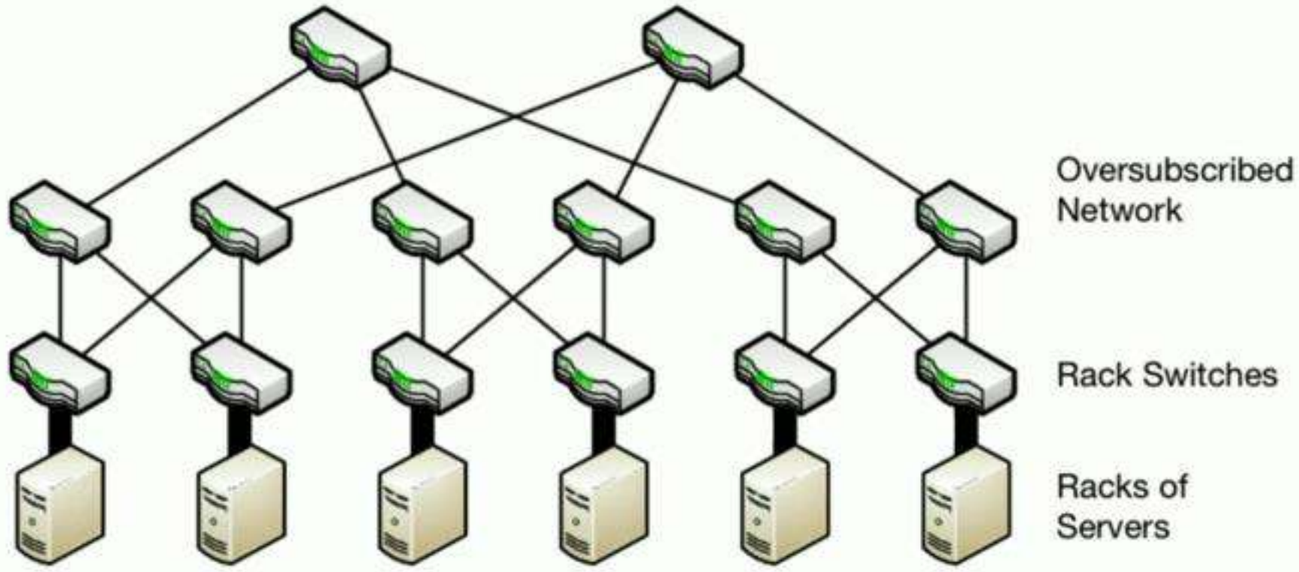
Graph-Centric Optimization

Evaluation

Conclusion

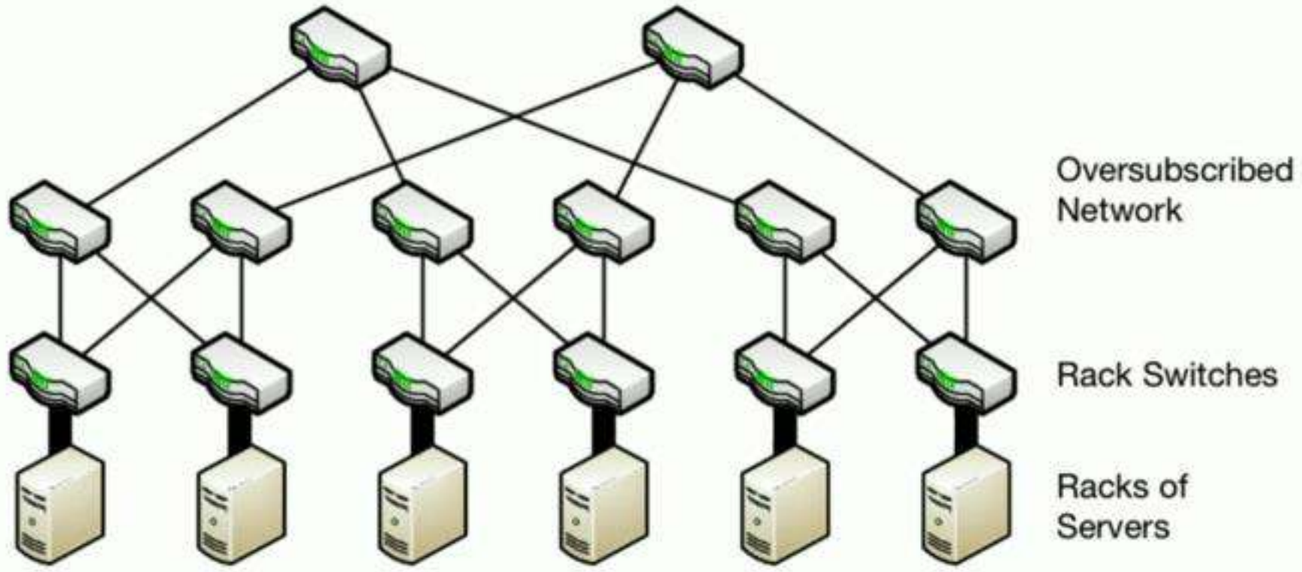
Data Center Background

- Design principles are consistent for most cloud data centers.



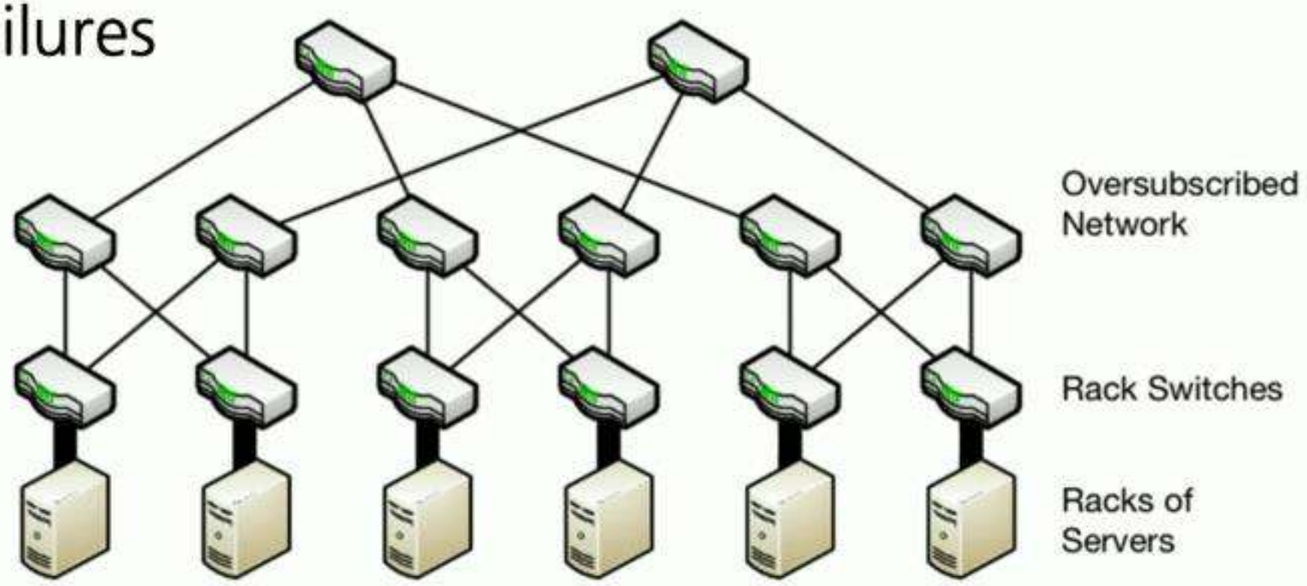
Data Center Background

- Design principles are consistent for most cloud data centers.
- The network is oversubscribed.



Data Center Background

- Design principles are consistent for most cloud data centers.
- The network is oversubscribed.
  - Intentionally designed
  - Link degradation/failures
  - Background traffic



Outline

Motivation & Proposal

Data Center Background

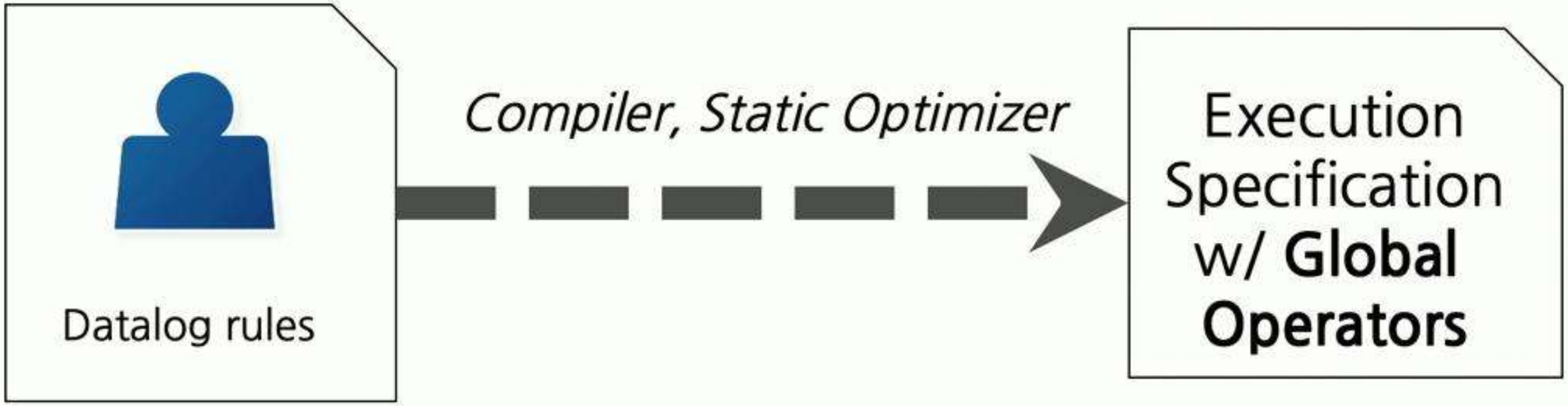
Data Center-Centric Optimization

Graph-Centric Optimization

Evaluation

Conclusion

Query Processing



Global Operators

Operators that are optimized at runtime to minimize communications.

- graph characteristics
- infrastructure characteristics

Global Operators

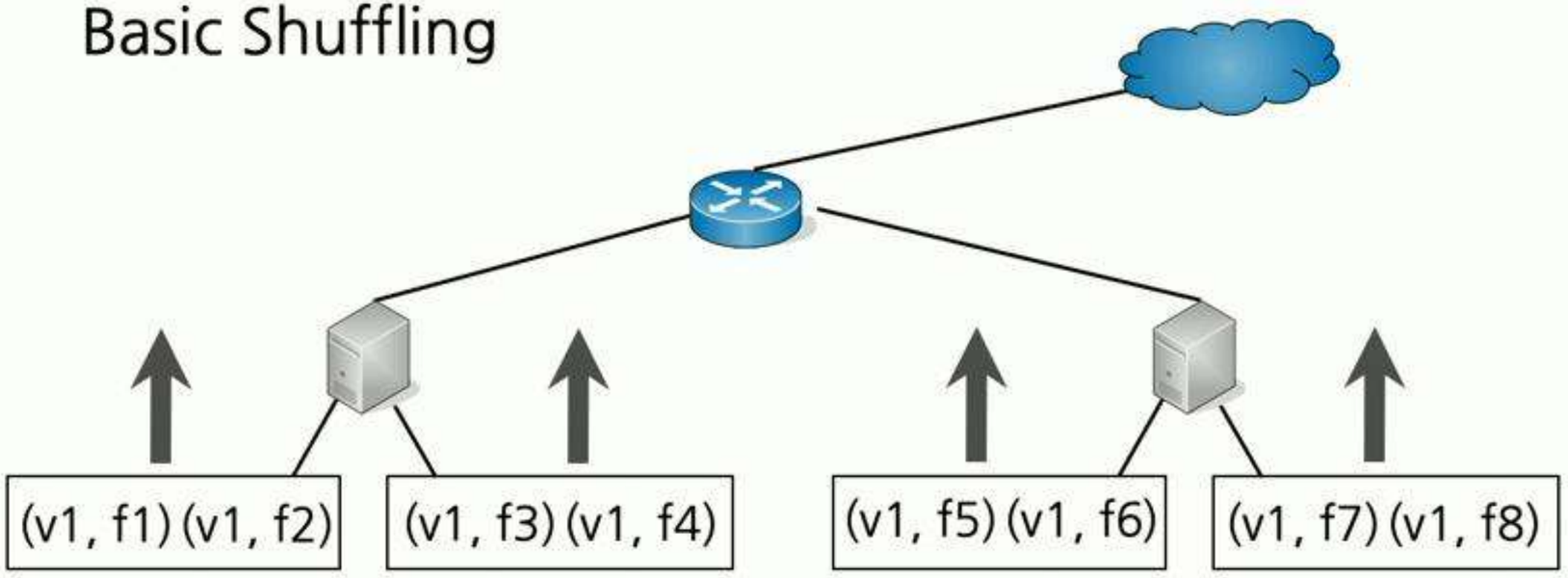
Operators that are optimized at runtime to minimize communications.

- graph characteristics
- infrastructure characteristics

- **SHUFF**: efficient shuffle operations
- **JOIN**: deduplication for binary joins
- **ROUT**: fine-grained join ordering for multi-way joins
- **AGG**: efficient aggregation evaluation

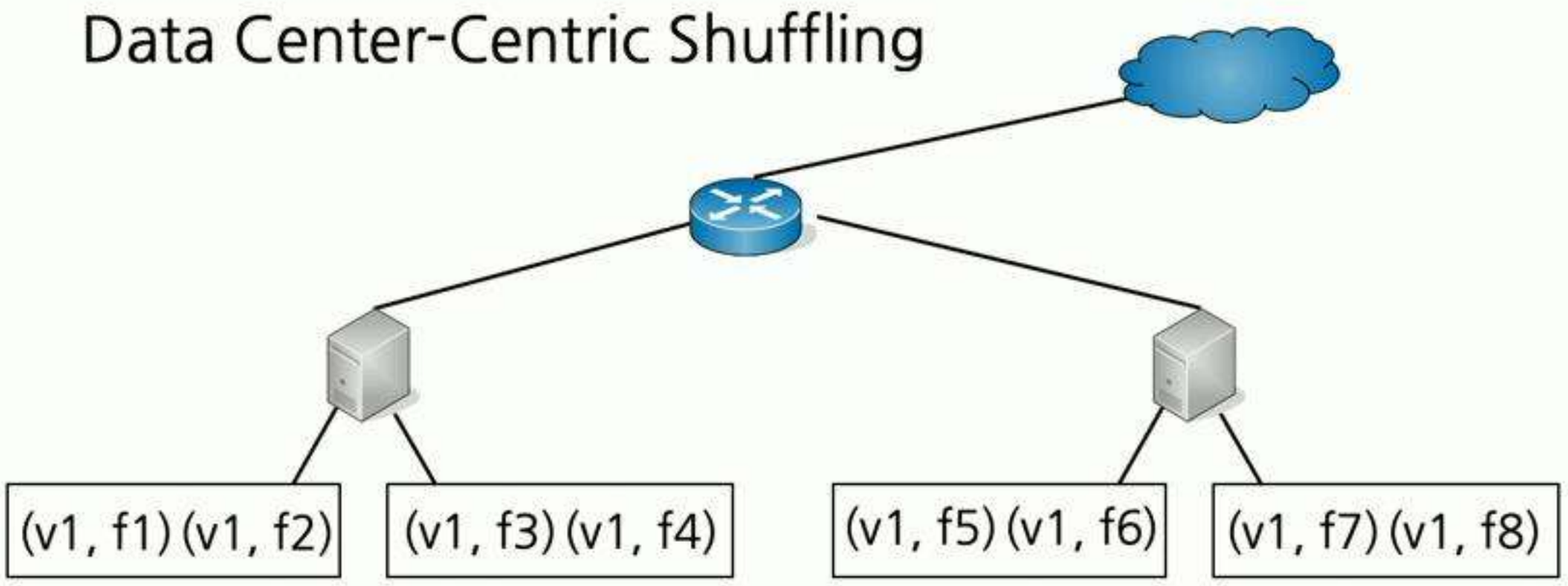
Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



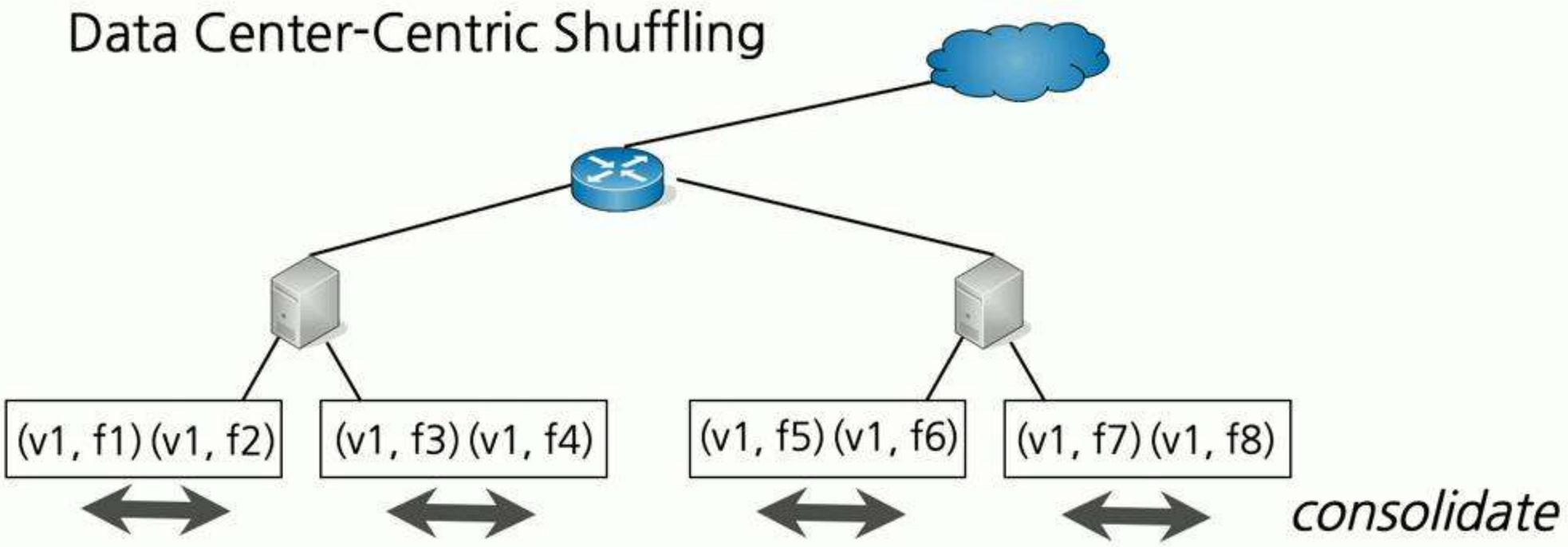
Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



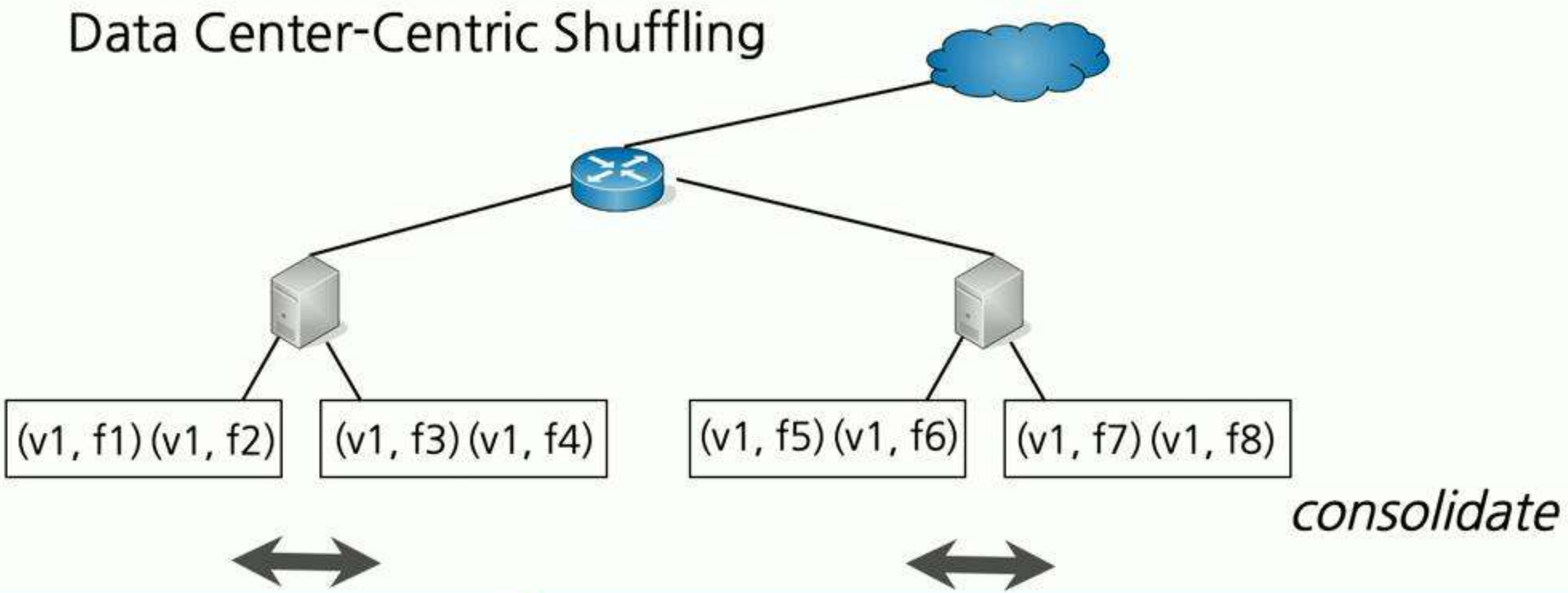
Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



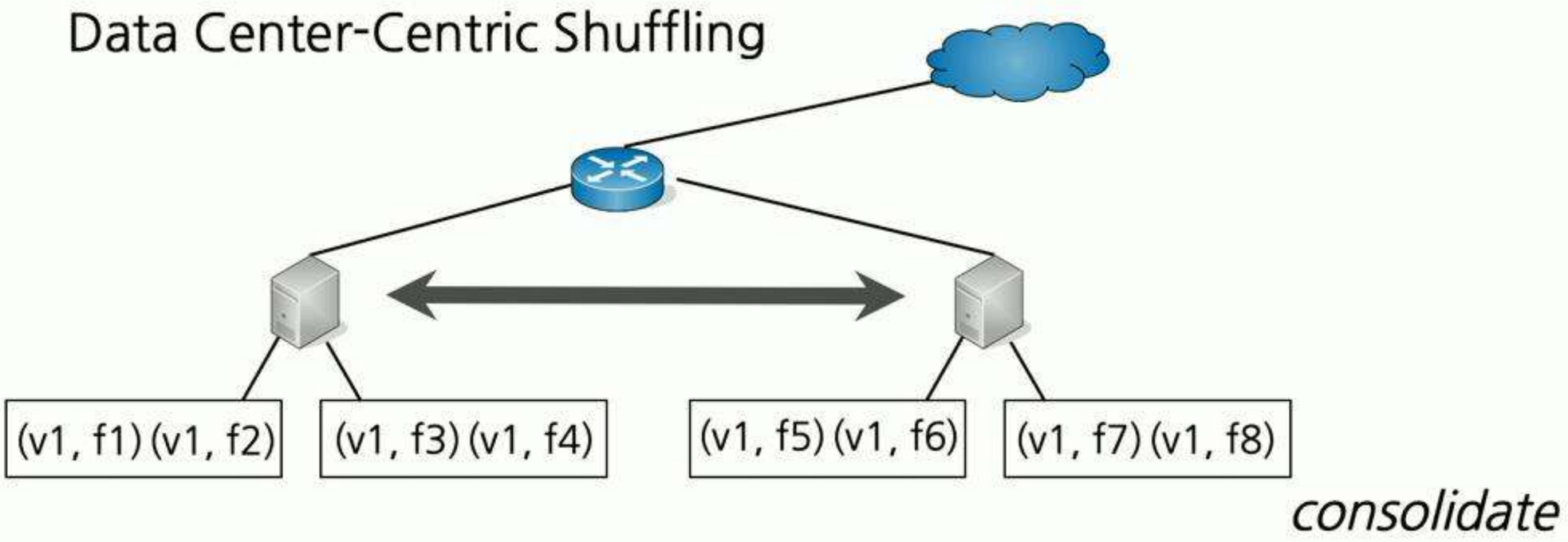
Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



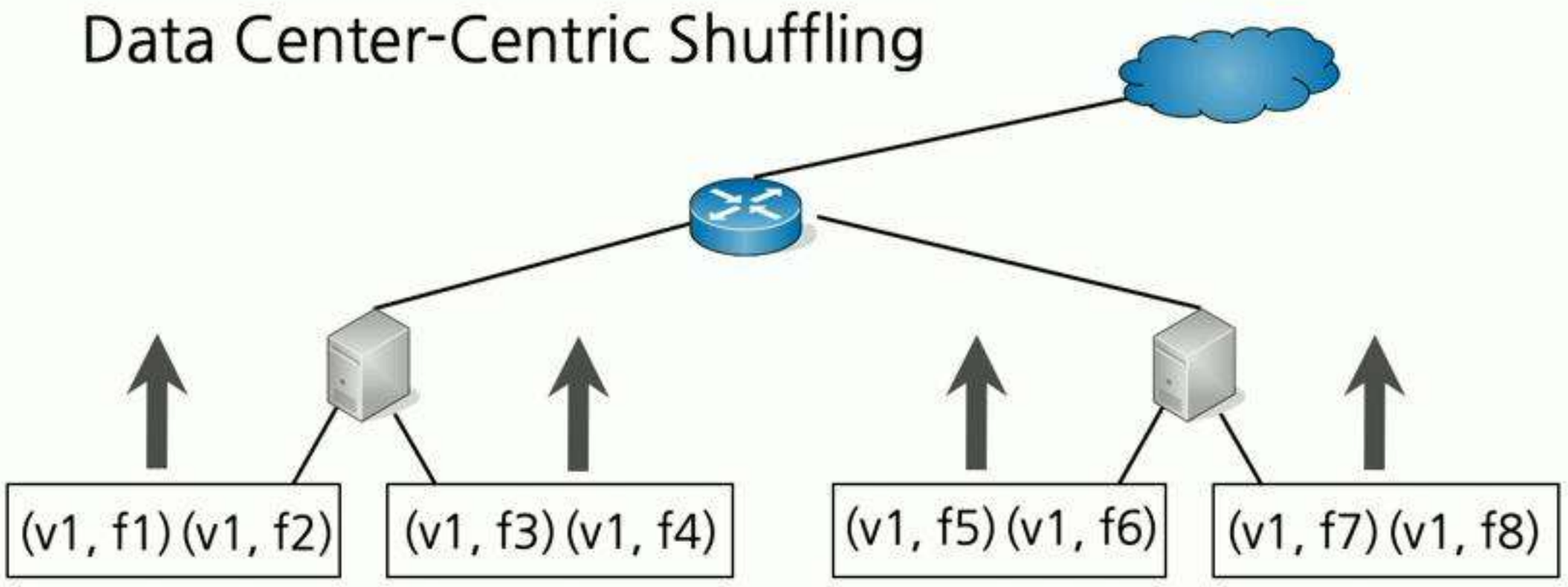
Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex

Optimization: Columnization & Compression

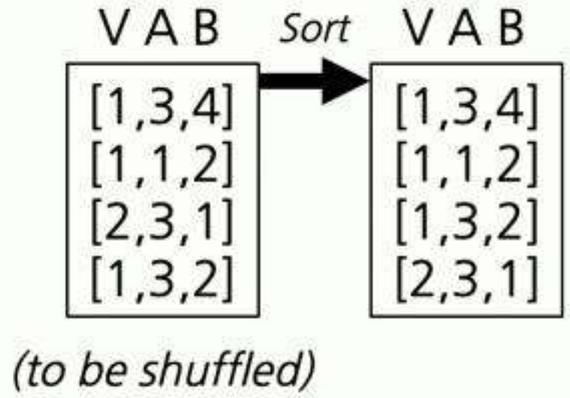
V	A	B
[1,3,4]		
[1,1,2]		
[2,3,1]		
[1,3,2]		

(to be shuffled)

Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex

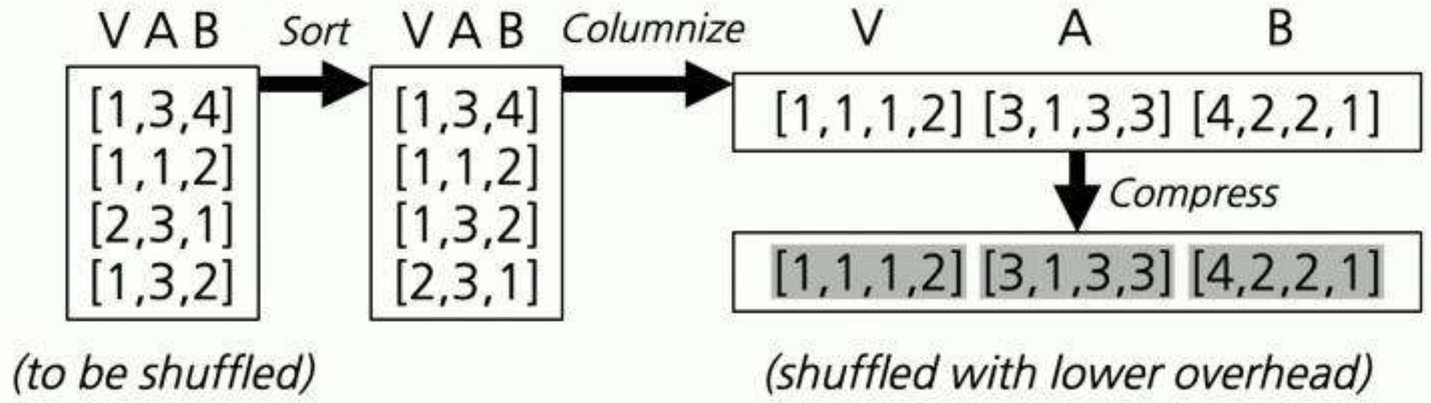
Optimization: Columnization & Compression



Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex

Optimization: Columnization & Compression



Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex

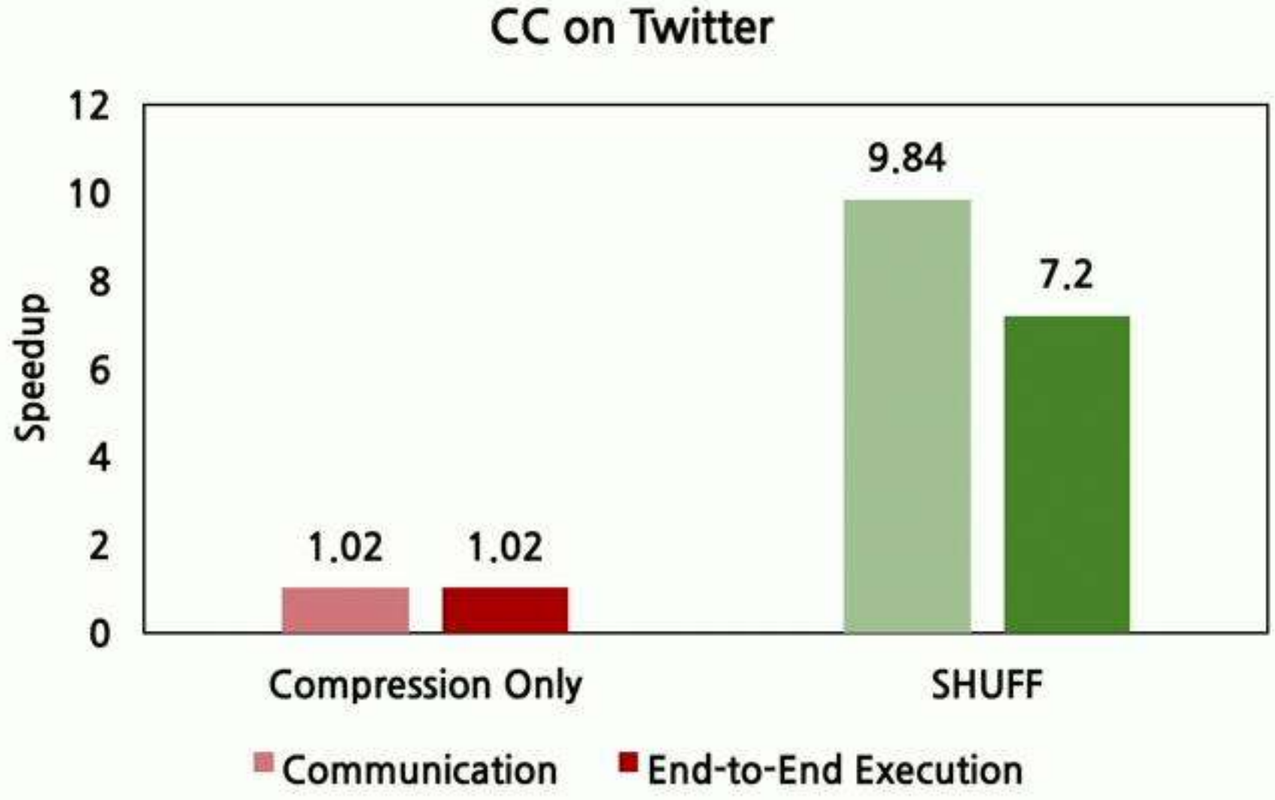
Optimization: Hierarchical Network Transfer

Three-Level Hierarchical Network Transfer

```
for each server S:
  workers_in_a_server = getWorkers(S)
  localShuffle(workers_in_a_server)
  consolidate messages
for each rack R:
  workers_in_a_rack = getWorkers(R)
  localShuffle(workers_in_a_rack)
  consolidate messages
globalShuffle(all_workers)
```

Data Center-Centric Optimization

SHUFF encompasses most network communication in GraphRex



Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Multi-Way Join

```
sg(A,B) :- e(X,A), sg(X,Y), e(Y,B)
```

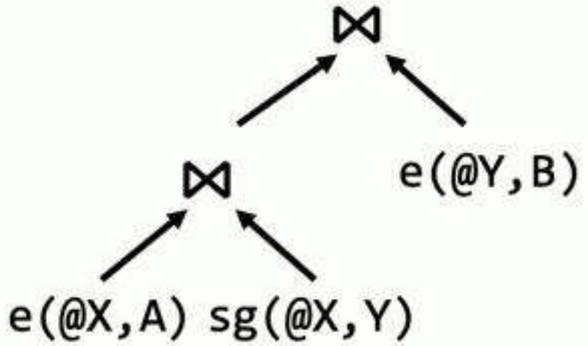
Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

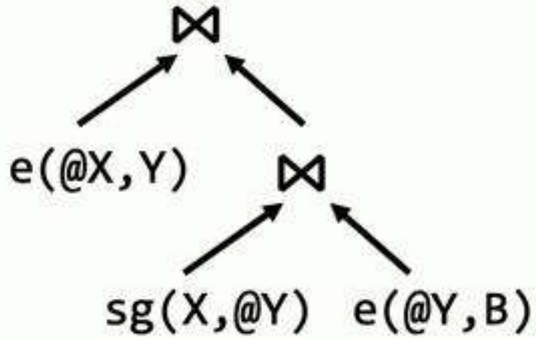
Multi-Way Join

```
sg(A,B) :- e(X,A), sg(X,Y), e(Y,B)
```

Order 1: from left to right



Order 2: from right to left

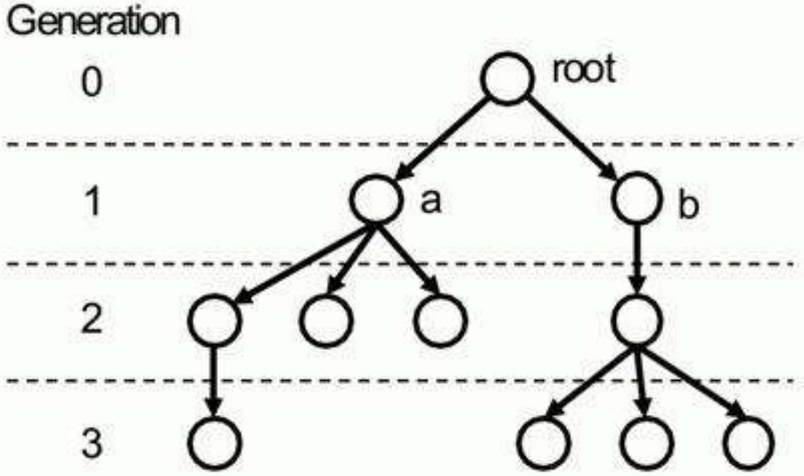


Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Multi-Way Join

```
sg(A,B) :- e(X,A), sg(X,Y), e(Y,B)
```



sg(a, b)

Order 1
[[e ⋈ sg] ⋈ e]

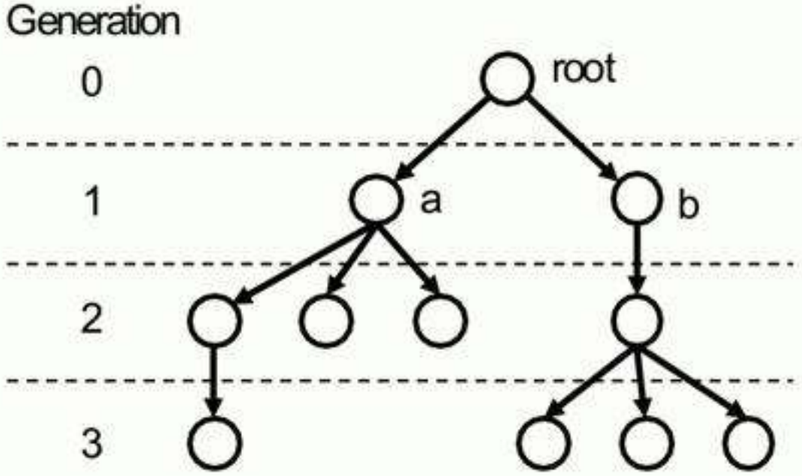
Order 2
[e ⋈ [sg ⋈ e]]

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Multi-Way Join

```
sg(A,B) :- e(X,A), sg(X,Y), e(Y,B)
```



sg(a, b)

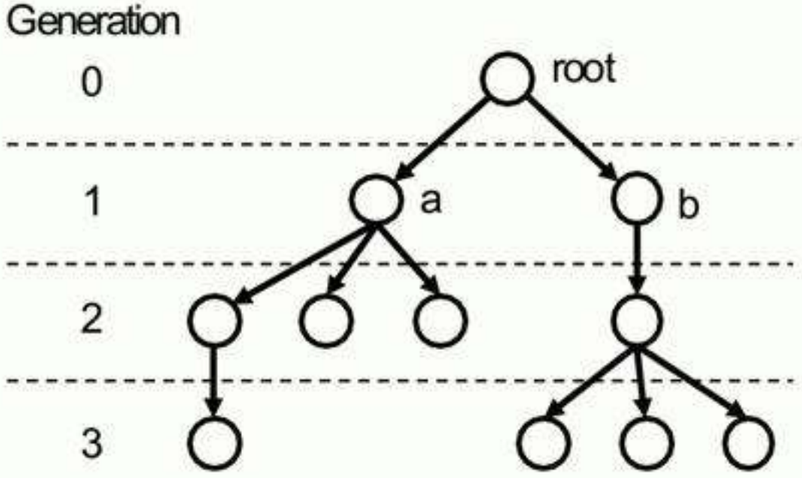
	Order 1 [[e ⋈ sg] ⋈ e]	**Order 2** [e ⋈ [sg ⋈ e]]
G2 Cost:	**3**	**1**

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Multi-Way Join

```
sg(A,B) :- e(X,A), sg(X,Y), e(Y,B)
```



sg(a, b)

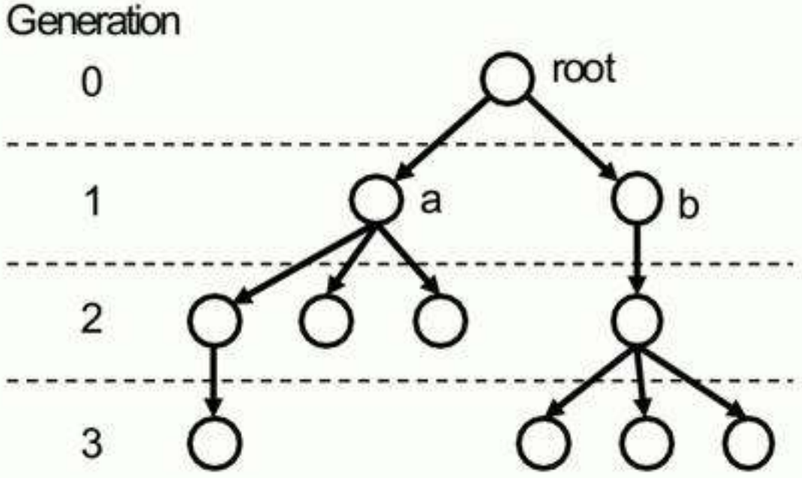
	Order 1 [[e ⋈ sg] ⋈ e]	Order 2 [e ⋈ [sg ⋈ e]]
G2 Cost:	**3**	**1**
G3 Cost:	**1**	**3**

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Multi-Way Join

```
sg(A,B) :- e(X,A), sg(X,Y), e(Y,B)
```



sg(a, b)

	Order 1 [[e ⋈ sg] ⋈ e]	**Order 2** [e ⋈ [sg ⋈ e]]
G2 Cost:	**3**	**1**
G3 Cost:	**1**	**3**
Total Cost:	**4**	**4**

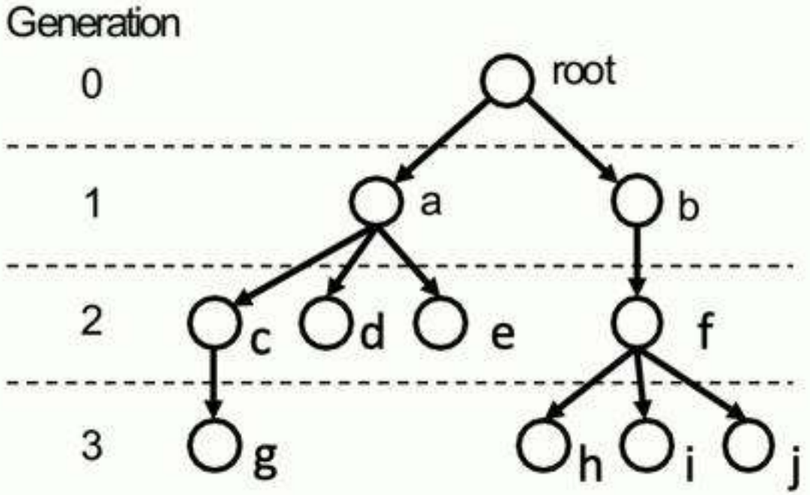
Order 1 and Order 2 are equally bad.
Power-law degree distribution makes them worse.

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Optimization: Adaptive Join Ordering

Enumerate all possible join orders for each newly generated tuple, and select the minimum-cost order.



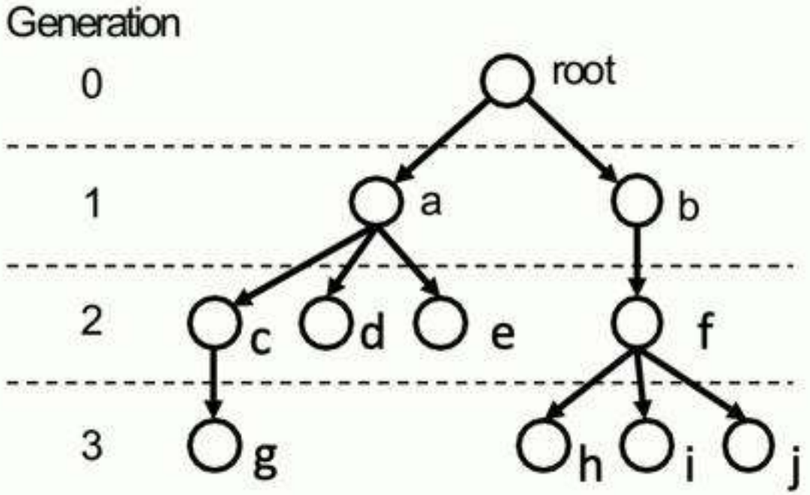
sg(a, @b)
[e ⋈ [sg ⋈ e]]

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Optimization: Adaptive Join Ordering

Enumerate all possible join orders for each newly generated tuple, and select the minimum-cost order.



sg(a, @b)
[e ⋈ [sg ⋈ e]]

sg(@c, f)
[[e ⋈ sg] ⋈ e]

sg(@d, f)
[[e ⋈ sg] ⋈ e]

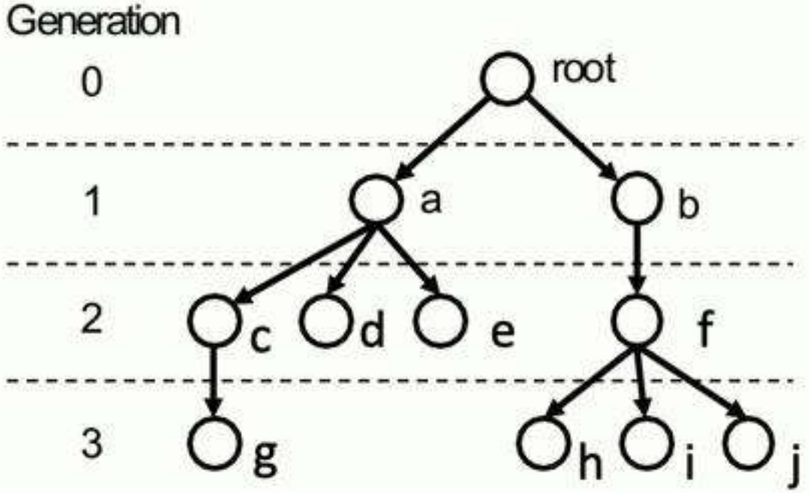
sg(@e, f)
[[e ⋈ sg] ⋈ e]

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Optimization: Adaptive Join Ordering

Enumerate all possible join orders for each newly generated tuple, and select the minimum-cost order.



sg(a, @b)
[e ⋈ [sg ⋈ e]]

sg(@c, f)
[[e ⋈ sg] ⋈ e]

sg(@d, f)
[[e ⋈ sg] ⋈ e]

sg(@e, f)
[[e ⋈ sg] ⋈ e]

sg(g, @h)
[e ⋈ [sg ⋈ e]]

sg(g, @i)
[e ⋈ [sg ⋈ e]]

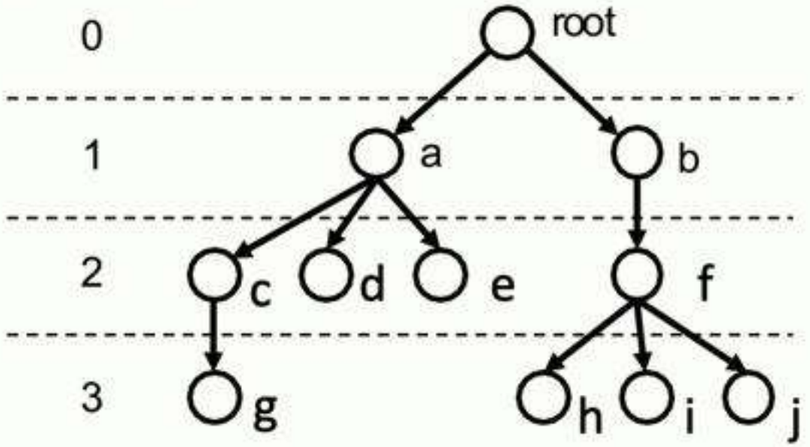
sg(g, @j)
[e ⋈ [sg ⋈ e]]

Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins

Optimization: Adaptive Join Ordering

Enumerate all possible join orders for each newly generated tuple, and select the minimum-cost order.



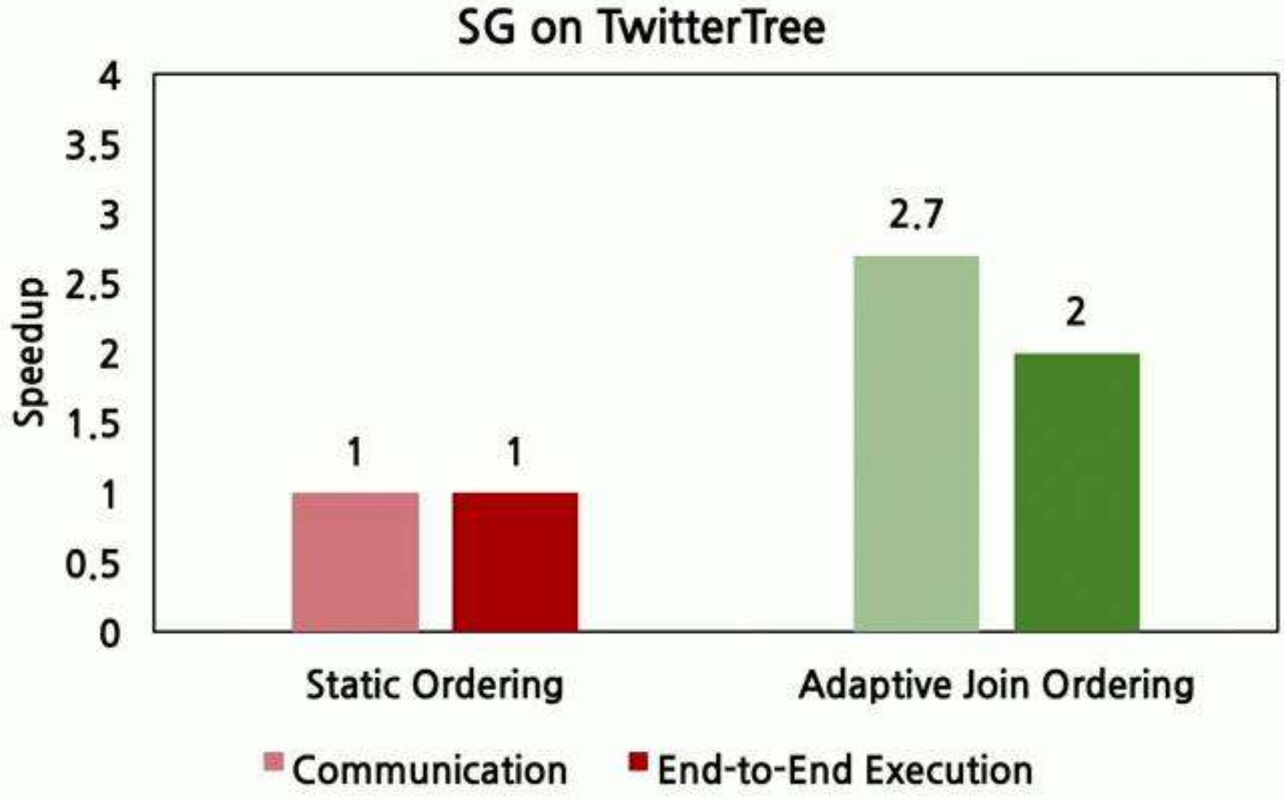
sg(a, b)

Adaptive Join Ordering

G2:	**1**
G3:	**1**
Total:	**2**

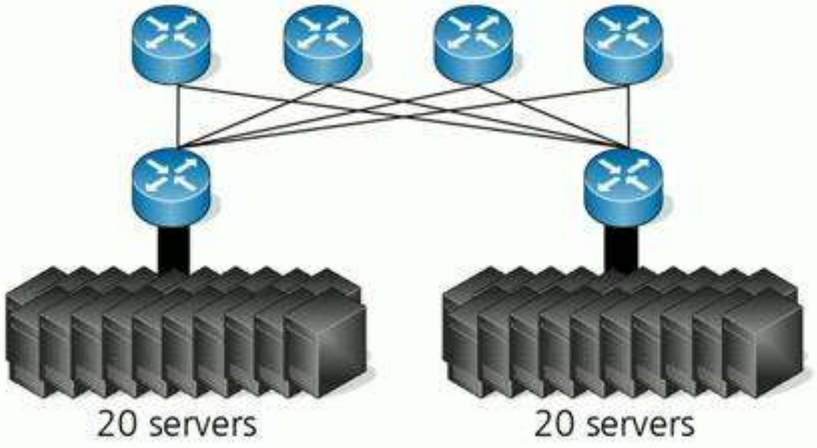
Graph-Centric Optimization

ROUT finds the optimal join order at tuple level for multi-way joins



Evaluation

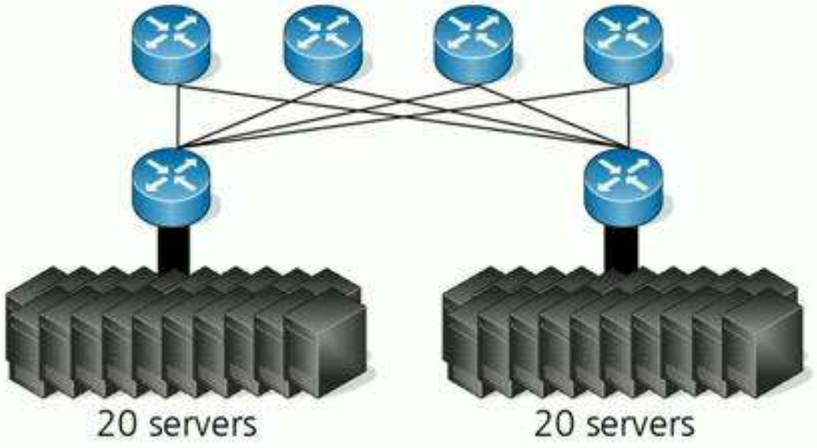
Setup



CPU: 1600 threads
Memory: 6.4 Terabytes
Network: 10 Gb/s links (5:1 oversubscription ratio)

Evaluation

Setup

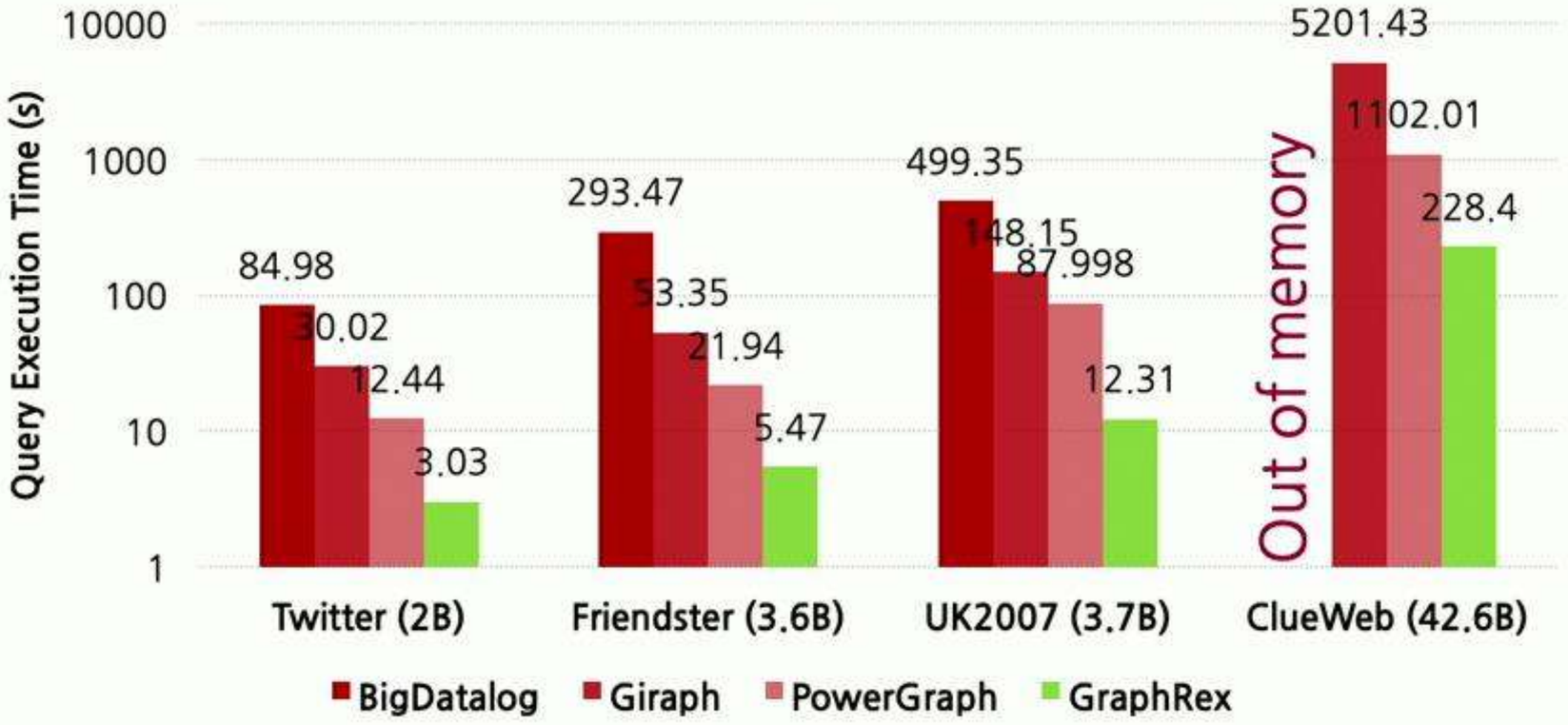


Graph	#Vertices	#Edges	Raw Size
Twitter	52.6 millions	2 billions	12 GB
Friendster	65.6 millions	3.6 billions	31 GB
UK2007	105.9 millions	3.7 billions	33 GB
ClueWeb	978.4 millions	42.6 billions	406 GB

CPU: 1600 threads
Memory: 6.4 Terabytes
Network: 10 Gb/s links (5:1 oversubscription ratio)

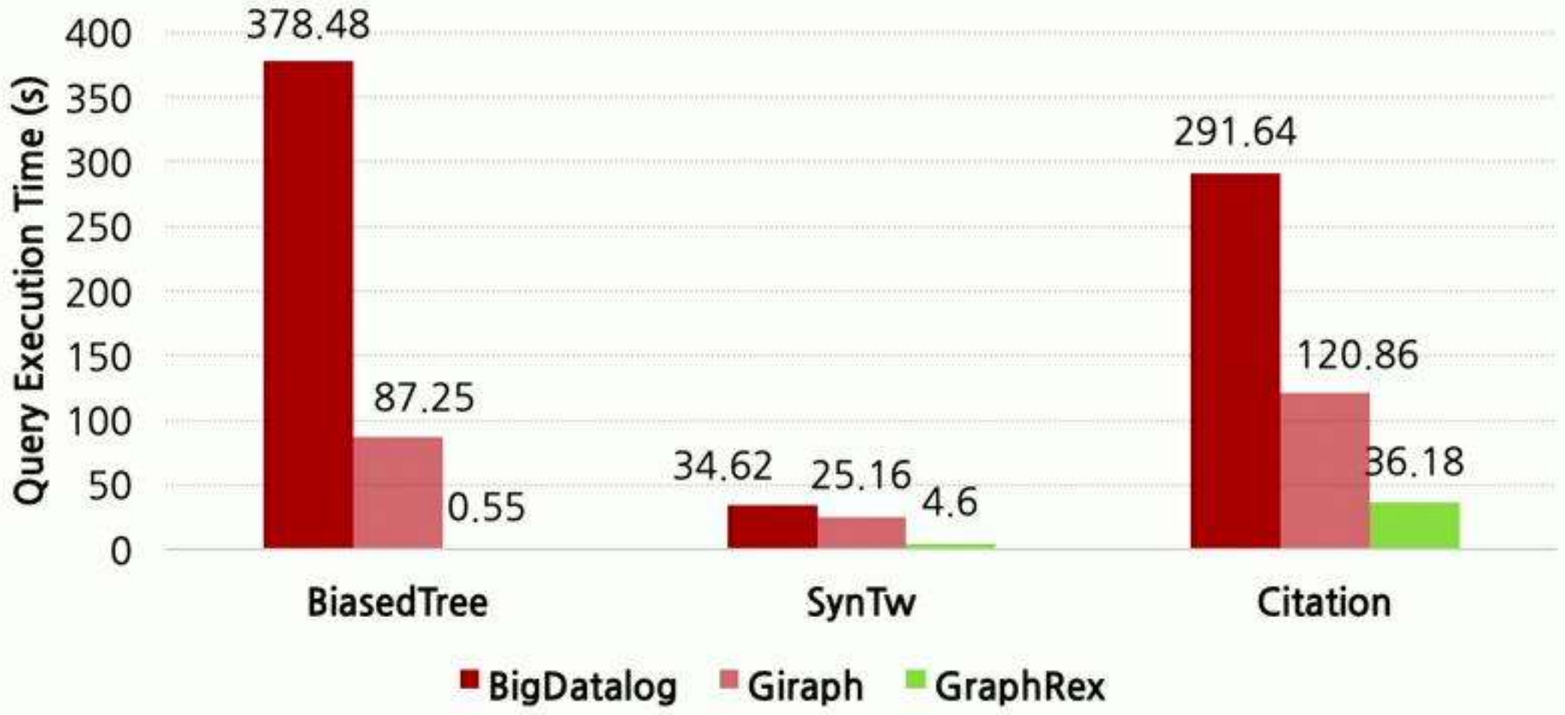
Evaluation

Overall Performance (SSSP): 4X - 54X Speedup



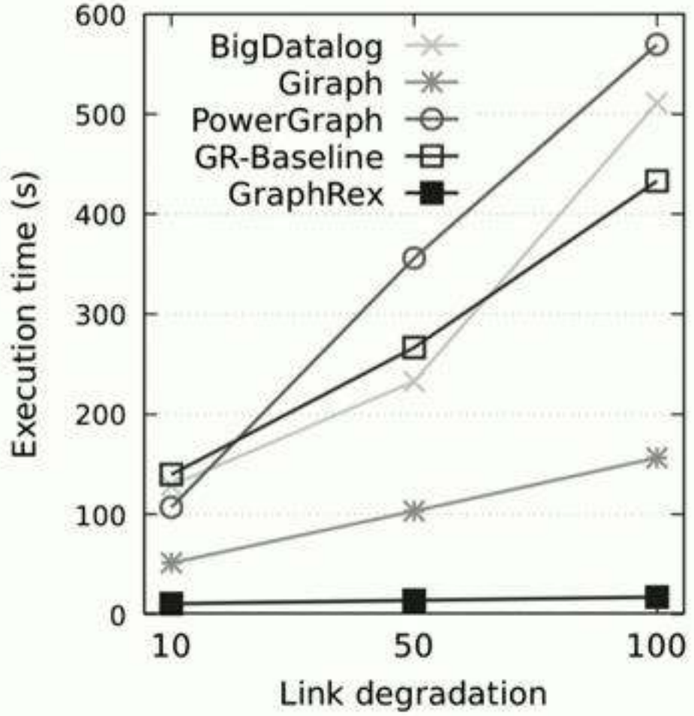
Evaluation

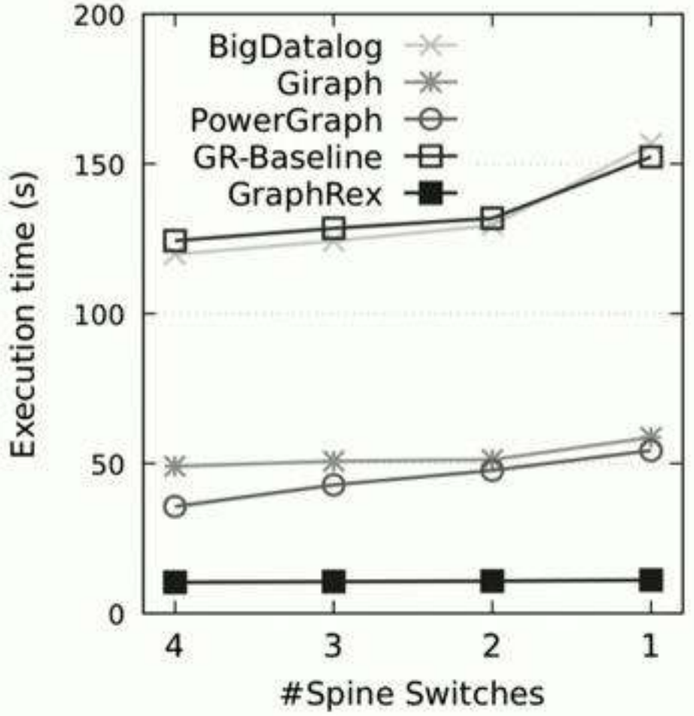
Multi-Way Join Performance: 3.3X - 688X Speedup



Evaluation

Robustness (Link Failures & Oversubscription)





Conclusion

- Communication bottlenecks the performance of large-scale graph processing.
- Achieving (1) ease of programming, (2) performance and (3) robustness needs the codesign between interface and optimizations.
- Efficient large-scale graph query processing requires the framework to be aware of data center infrastructure and graph data characteristics.

Future Work

- Query graph streams.

- Resource changes are normal and significant, especially prominent in streaming processing (long running).
- Failures can happen during query execution.

- No fixed (early-binding) plan is optimal.
- Optimizations have to consider multiple dimensions.
  - Workloads
  - Infrastructure
  - Random changes: both workloads and infrastructure

Thank you!

Questions?

Future Work

- Query graph streams.

- Resource changes are normal and significant, especially prominent in streaming processing (long running).
- Failures can happen during query execution.

- No fixed (early-binding) plan is optimal.
- Optimizations have to consider multiple dimensions.
  - Workloads
  - Infrastructure
  - Random changes: both workloads and infrastructure