

Global analytics in the face of bandwidth and regulatory constraints

Ashish Vulimiri^u
Thomas Jungblut^m

Carlo Curino^m
Jitu Padhye^m

Brighten Godfrey^u
George Varghese^m






^uUIUC

^mMicrosoft

Massive data volumes

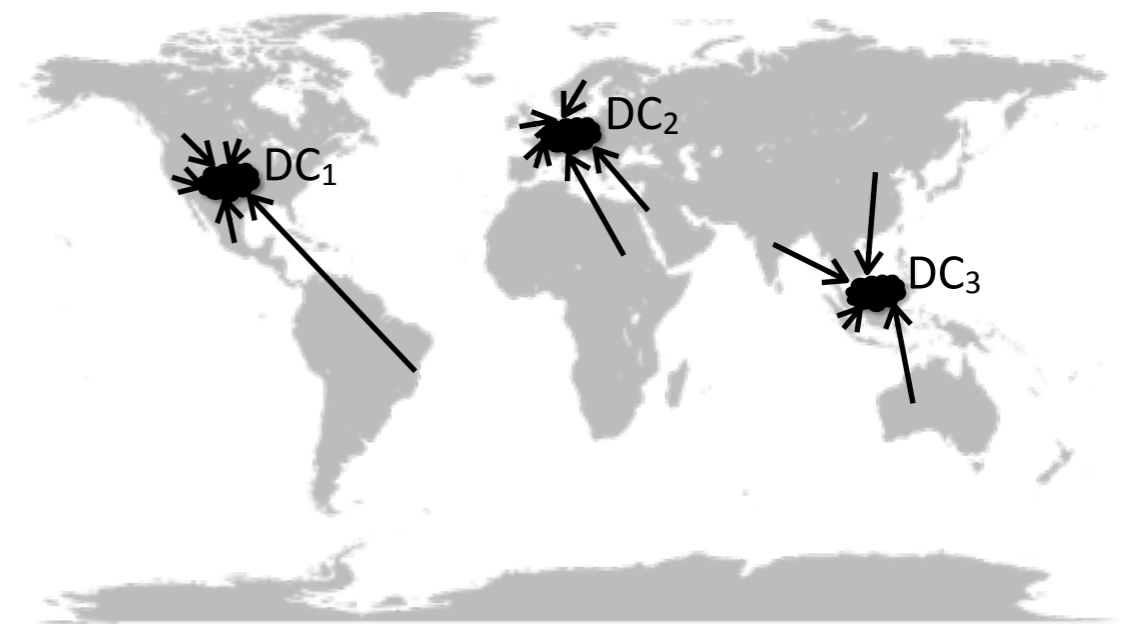
	Facebook	600 TB/day
	Twitter	100 TB/day
	Microsoft	10s TB/day
	LinkedIn	10 TB/day
	Yahoo!	10 TB/day

Massive data volumes

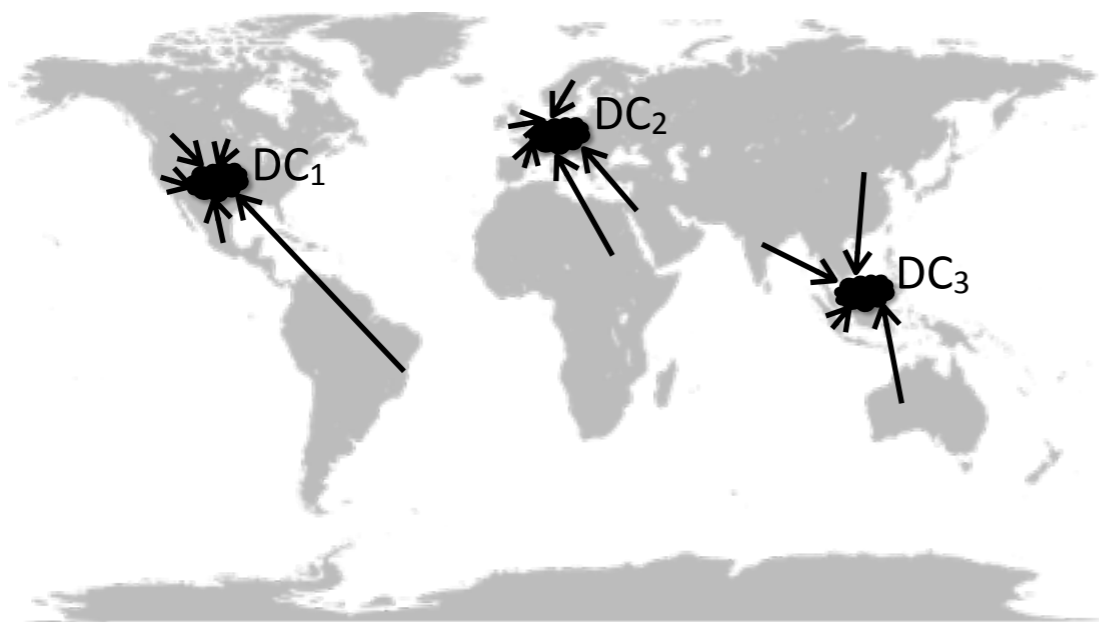
 Facebook	600 TB/day
 Twitter	100 TB/day
 Microsoft	10s TB/day
 LinkedIn	10 TB/day
 Yahoo!	10 TB/day

Use cases:

- User activity logs
- Monitoring remote infrastructures
- ...

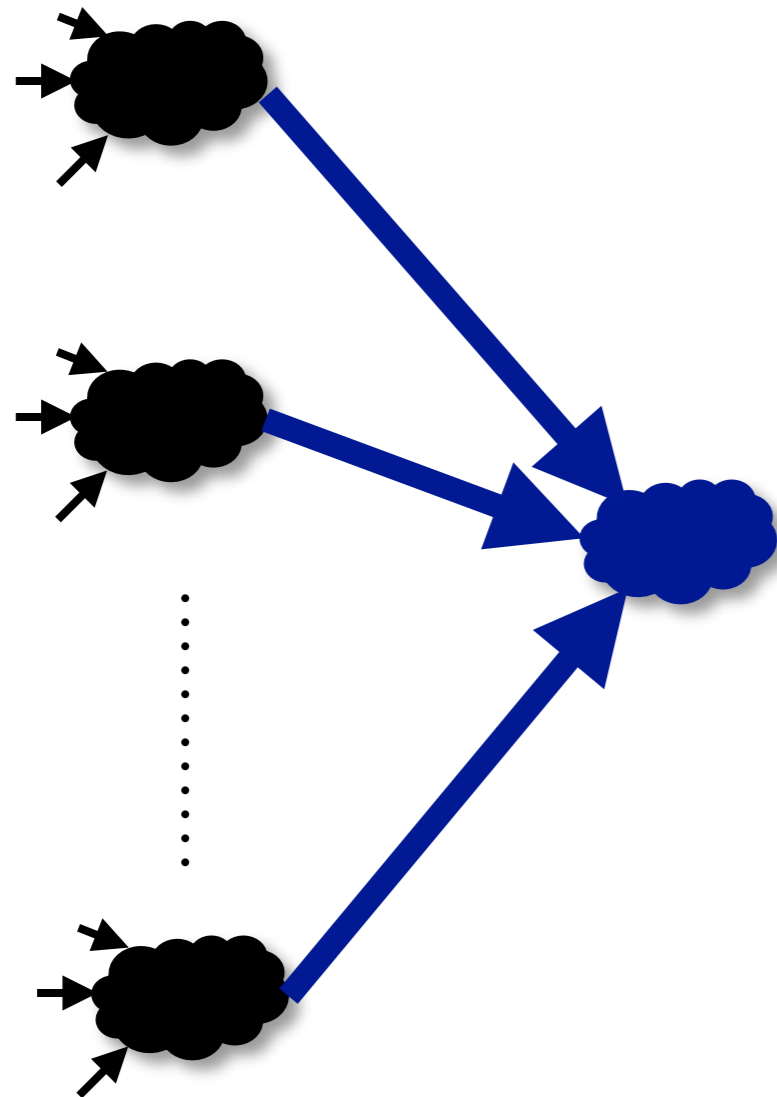


Collected across several data centers for low user latency





SQL analytics across geo-distributed data to extract insight



current solution: centralize

- copy all data to central data center
- run all queries there

10s-100s TB/day
up to 10s of DCs

current solution: copy all data to central DC, run all analytics there

Centralized approach is inadequate

1. Consumes scarce, expensive cross-DC bandwidth

current solution: copy all data to central DC, run all analytics there

Centralized approach is inadequate

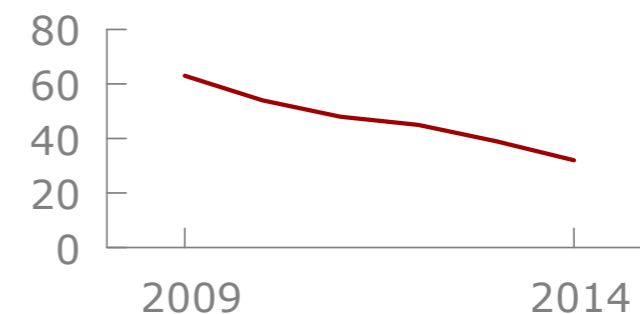
1. Consumes scarce, expensive cross-DC bandwidth

rising costs

external network is
fastest rising DC cost

slowing growth

Internet capacity growth (%)



scarce capacity

total Internet capacity \ll some DCs' internal bisection b/w

100 Tb/s 1000Tb/s

recognized concern

several other efforts to
reduce wide-area traffic
e.g. SWAN, B4

current solution: copy all data to central DC, run all analytics there

Centralized approach is inadequate

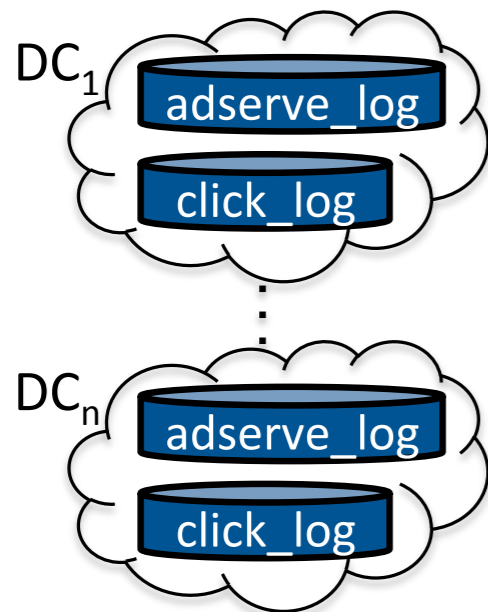
1. Consumes scarce, expensive cross-DC bandwidth
2. Incompatible with sovereignty concerns
 - Many countries considering restricting moving citizens' data
 - Could render centralization impossible
 - Speculation: *derived* information might still be acceptable

current solution: copy all data to central DC, run all analytics there

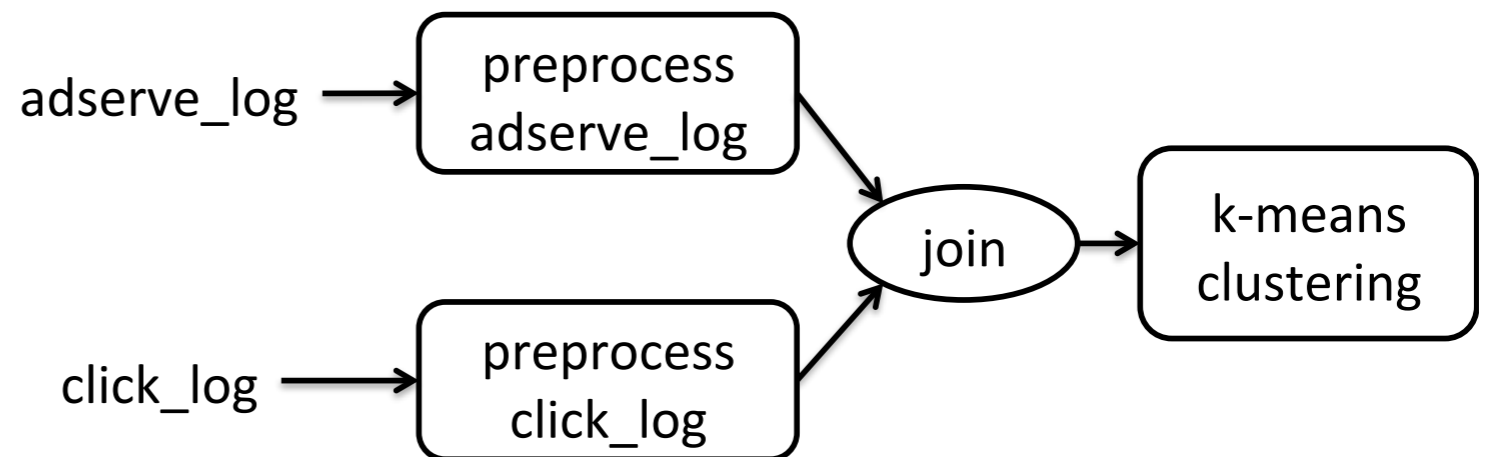
Centralized approach is inadequate

1. Consumes scarce, expensive cross-DC bandwidth
2. Incompatible with sovereignty concerns

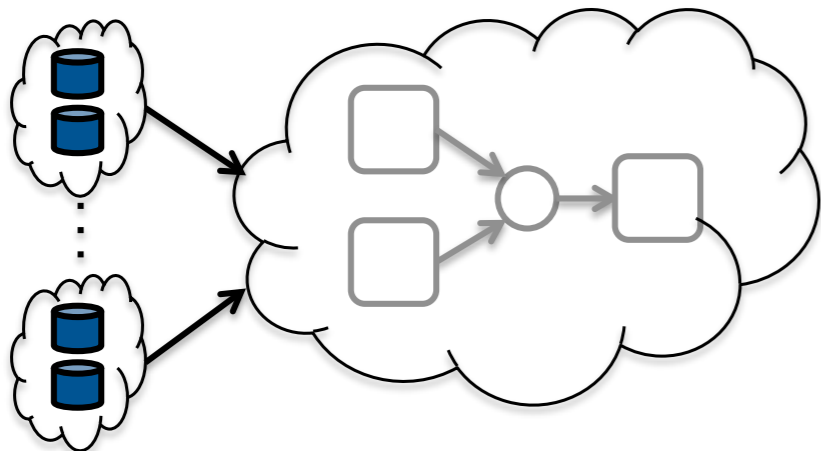
Geo-distributed SQL analytics



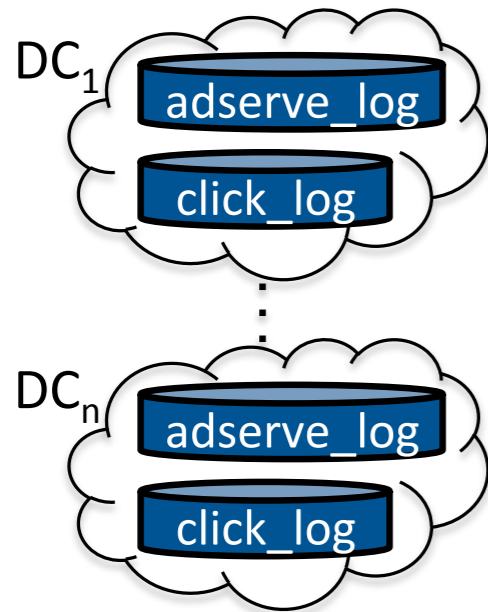
SQL query:



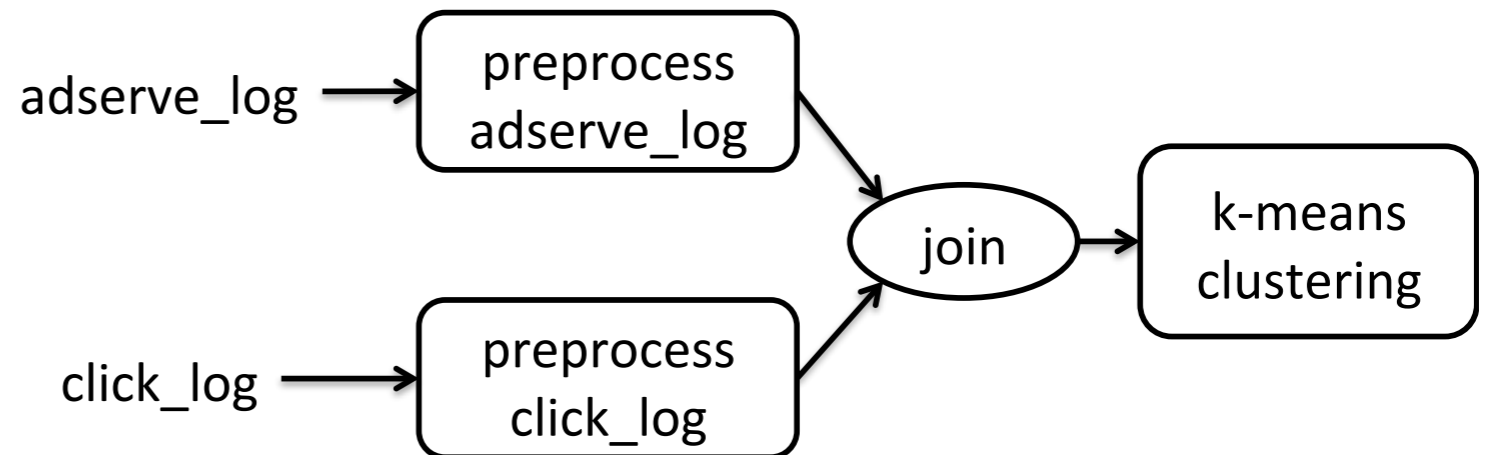
Centralized execution: 10 TB/day



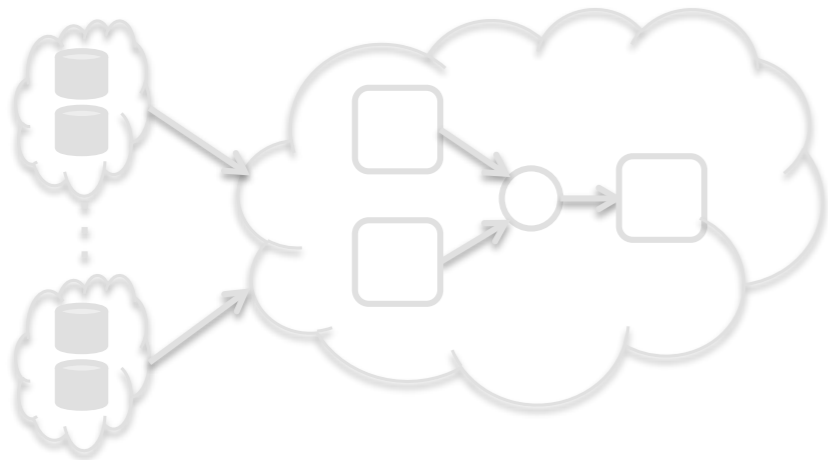
Geo-distributed SQL analytics



SQL query:

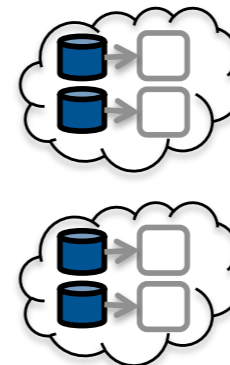


Centralized execution: **10 TB/day**

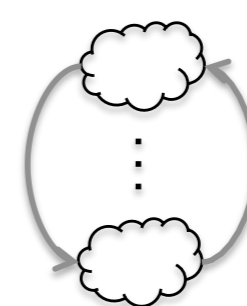


Distributed execution: **0.03 TB/day**

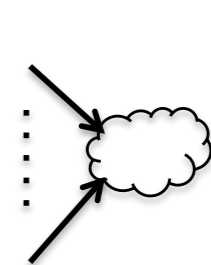
t = 0
push down
preprocess



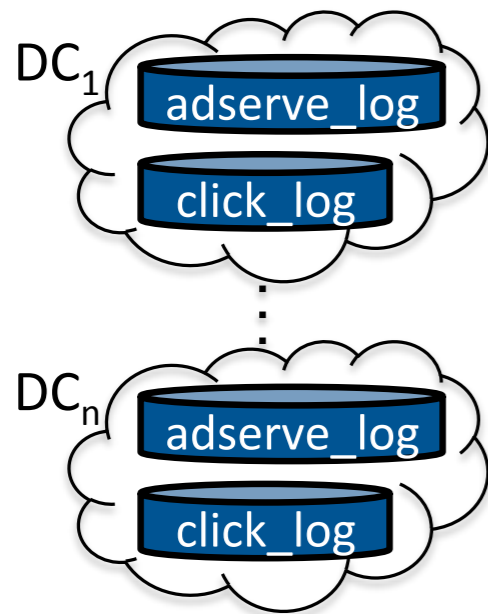
t = 1
distributed
semi-join



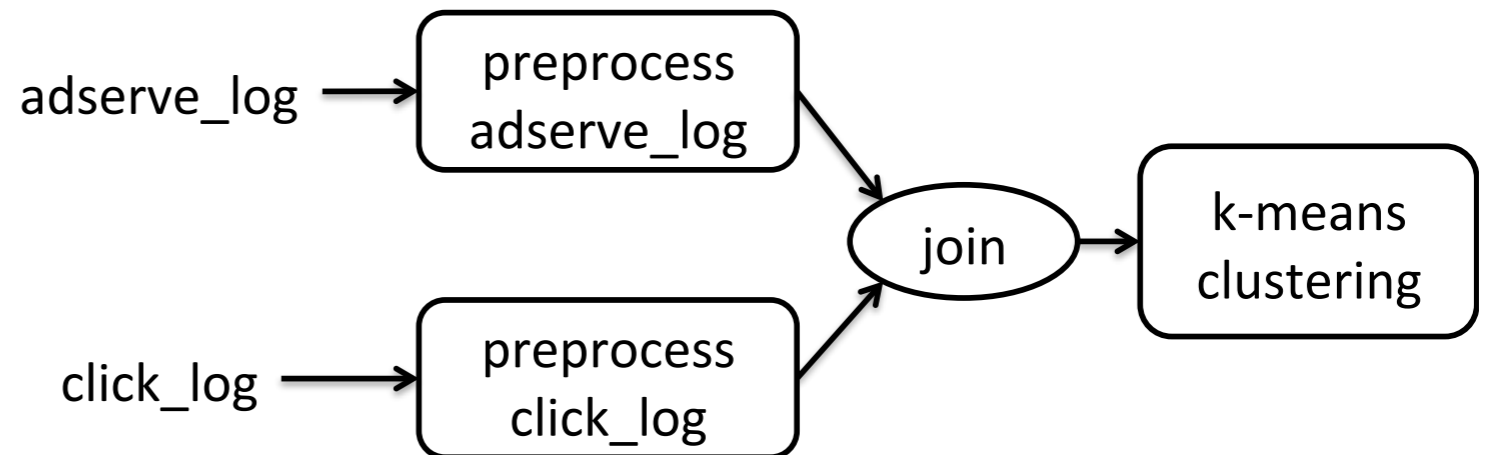
t = 2
centralized
k-means



Geo-distributed SQL analytics



SQL query:



Centralized execution: **10 TB/day**

Distributed execution: **0.03 TB/day**



Geo-distributed SQL analytics

Optimizations: synthesize and extend ideas from

- Parallel and distributed databases
- Distributed systems

... as well as novel techniques of our own

Common thread: revisit classical database problems from
networking perspective

PROBLEM DEFINITION

Requirements

Possible challenges to address

Bandwidth

Sovereignty

Fault-tolerance

Latency

Consistency

We target the **batch analytics** dominant in organizations today

Key characteristics

1. Support full relational model
2. No control over data partitioning
 - Dictated by external factors, typically end-user latency
3. Cross-DC bandwidth is scarcest resource by far
 - CPU, storage etc within data centers are relatively cheap
4. Unique constraints
 - Heterogeneous bandwidth costs/capacities
 - Sovereignty
5. Bulk of load comes from ~stable recurring workload
 - Consistent with production logs

Problem statement

Given: data born distributed across DCs a certain way

Goal: support SQL analytics on this data

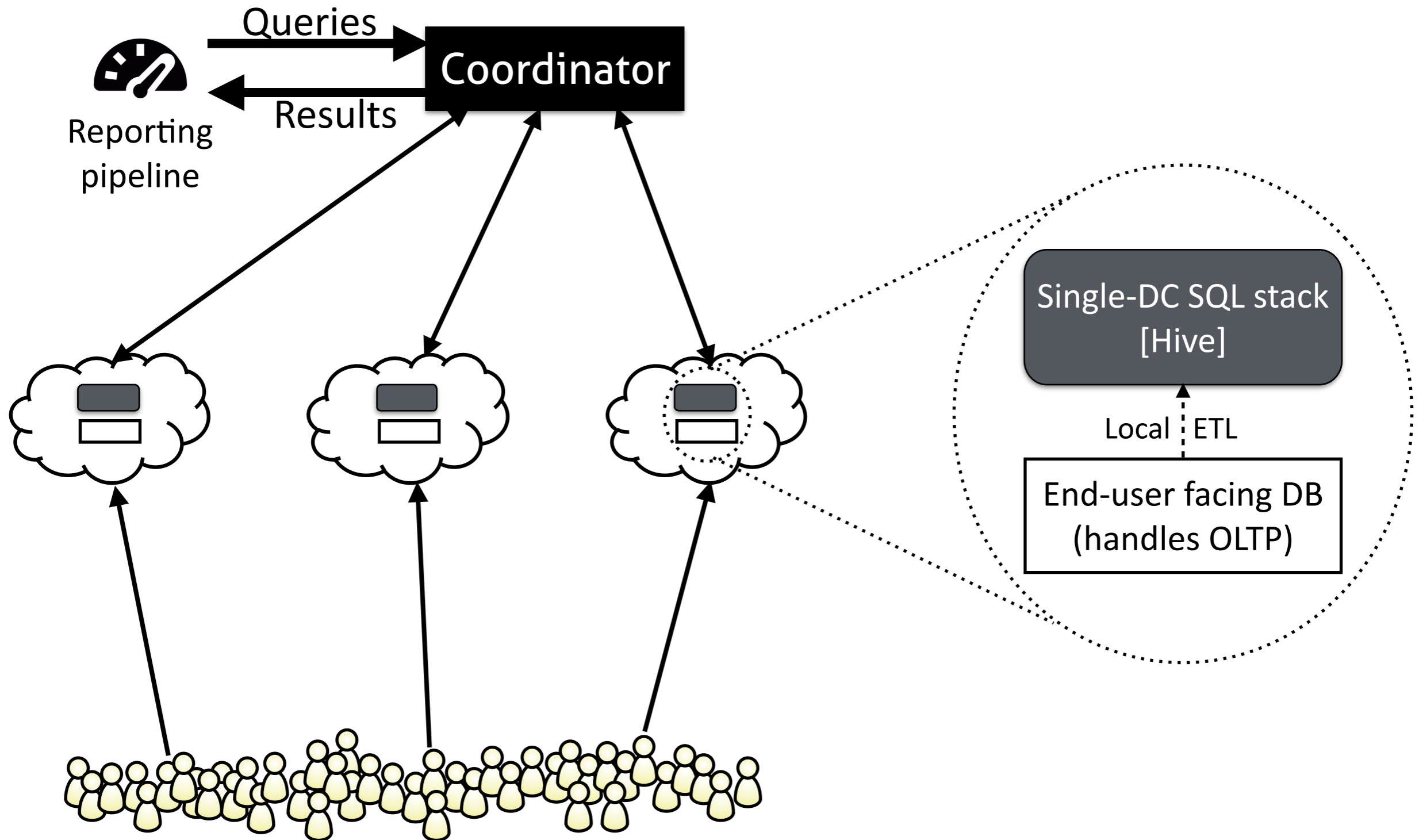
- Minimize bandwidth cost
- Handle fault-tolerance, sovereignty constraints

System will handle arbitrary queries at runtime

- But will be tuned to optimize known ~stable recurring workload

OUR APPROACH

Basic Architecture



Optimizations

Function-specific

SQL-aware
workload planning

Runtime
data transfer reduction



semantic
level

Optimizations

3. Function-specific

2. SQL-aware

1. Runtime

Optimizations

3. Function-specific

2. SQL-aware

1. Runtime

1. Runtime data transfer optimization

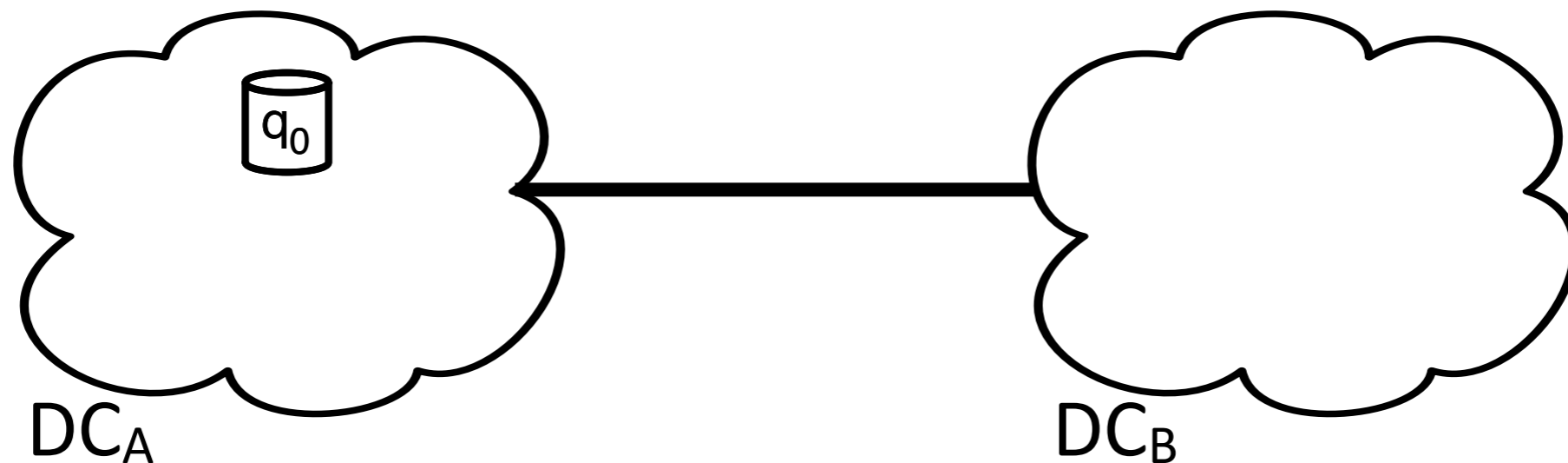
In our setting

- CPU, storage, ... within data centers is cheap
- Cross-DC bandwidth is the expensive resource

Trade off CPU, storage for bandwidth reduction

1. Runtime data transfer optimization

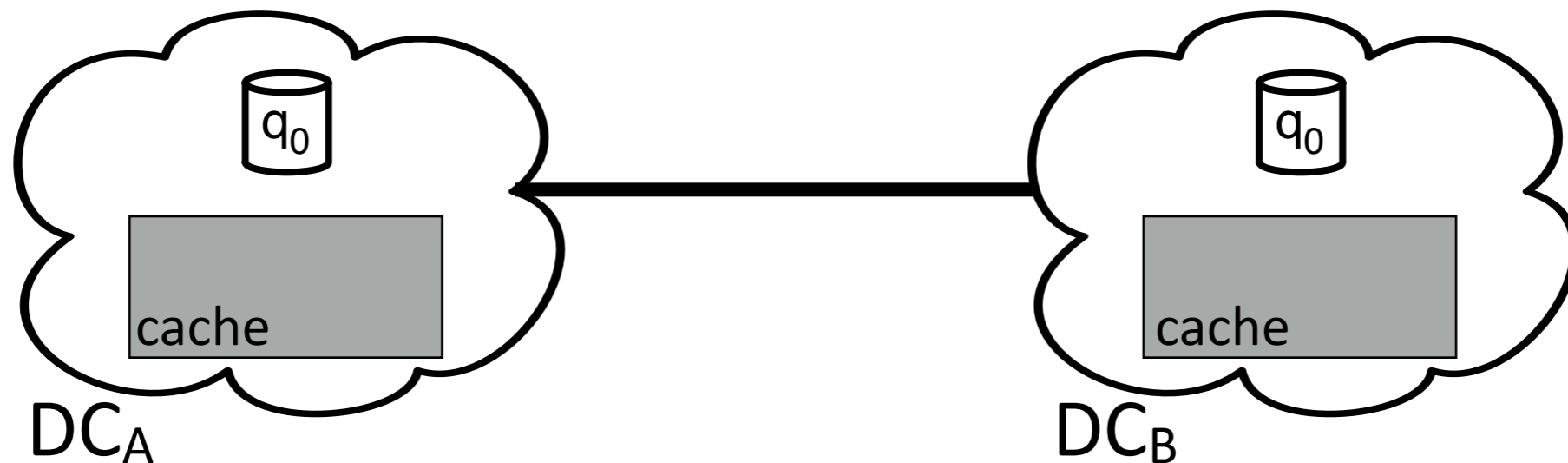
aggressively cache all intermediate output



$t = 0$ DC_B asks DC_A for results of subquery q

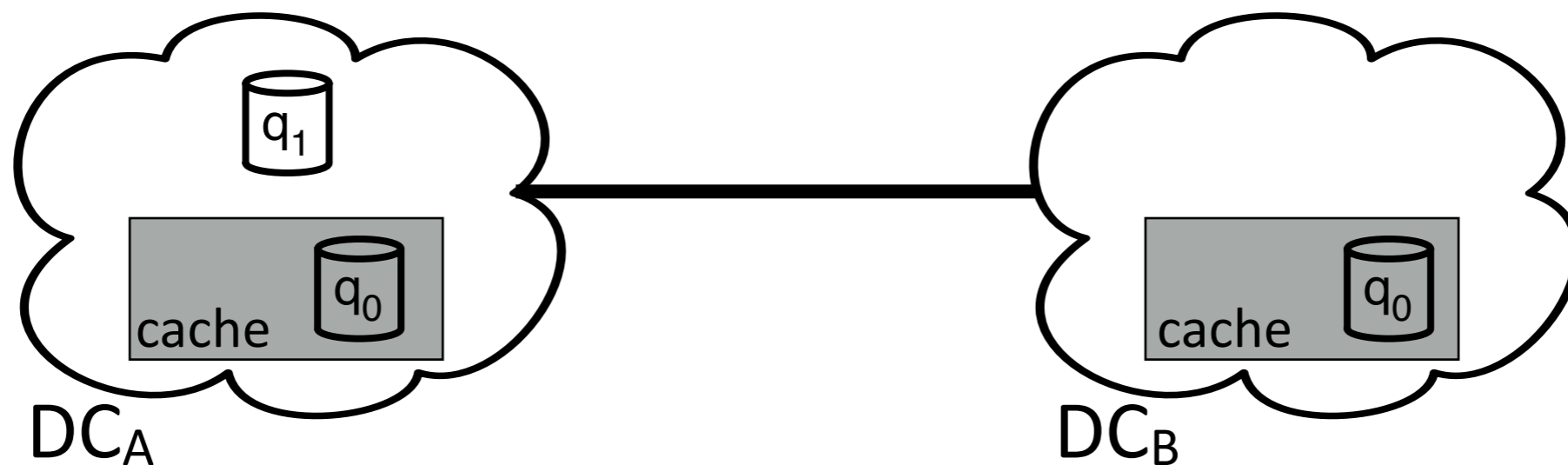
1. Runtime data transfer optimization

aggressively cache all intermediate output



1. Runtime data transfer optimization

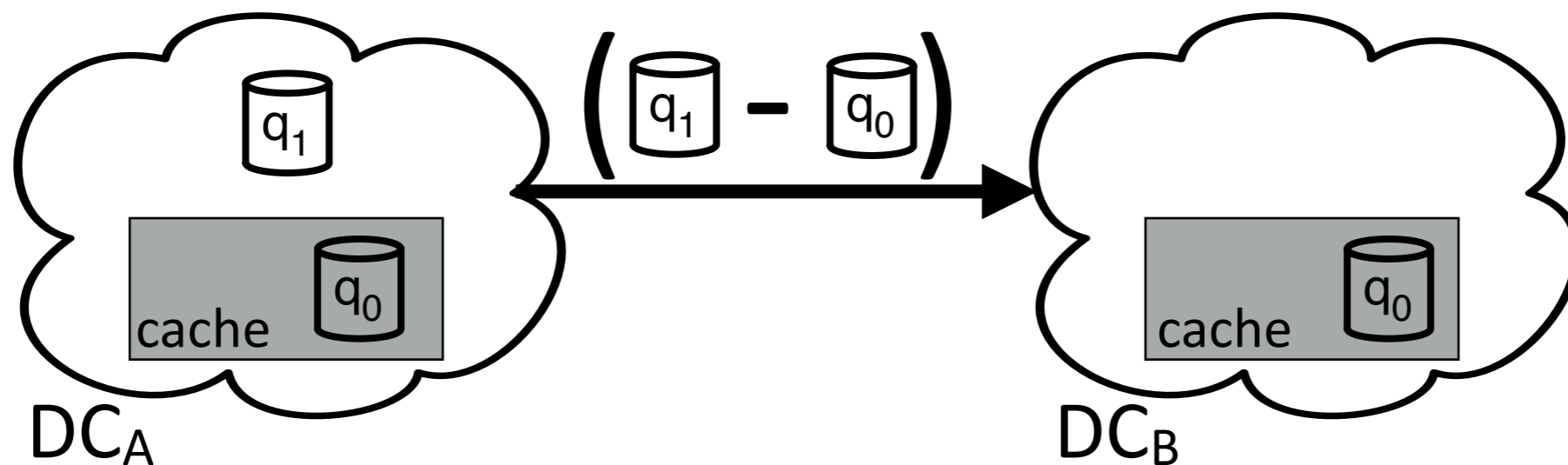
aggressively cache all intermediate output



$t = 1$ DC_B asks DC_A for results of subquery q again

1. Runtime data transfer optimization

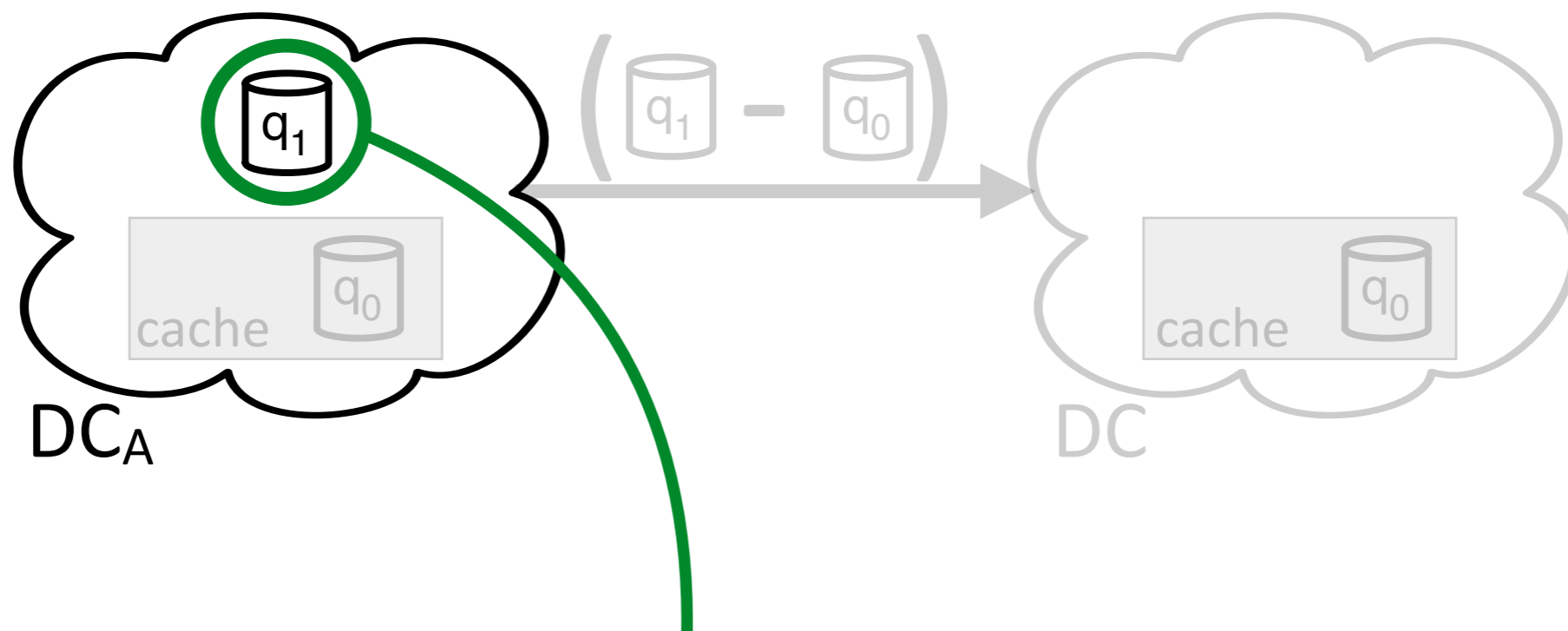
aggressively cache all intermediate output



$t = 1$ DC_B asks DC_A for results of subquery q again

1. Runtime data transfer optimization

aggressively cache all intermediate output



recompute q_1 from scratch

- not using caching to save latency, CPU
- only bandwidth

1. Runtime data transfer optimization

aggressively cache all intermediate output

Caching helps not only when same query arrives repeatedly

... but also when different queries have common sub-operations

e.g. 6x data transfer reduction in TPC-CH

1. Runtime data transfer optimization

aggressively cache all intermediate output

Caching helps not only when same query arrives repeatedly

... but also when different queries have common sub-operations

e.g. 6x data transfer reduction in TPC-CH

Database parallel: caching \approx view materialization

- Caching is a low-level, mechanical form of view maintenance
- + Works for arbitrary computations, including arbitrary UDFs
- Uses more CPU, storage
- Can miss opportunities

Optimizations

3. Function-specific

2. SQL-aware

1. Runtime

Optimizations

3. Function-specific

2. SQL-aware

1. Runtime

2. SQL-aware workload planning

Given

- Stable workload (set of queries)
- Fault-tolerance and sovereignty constraints

Jointly optimize

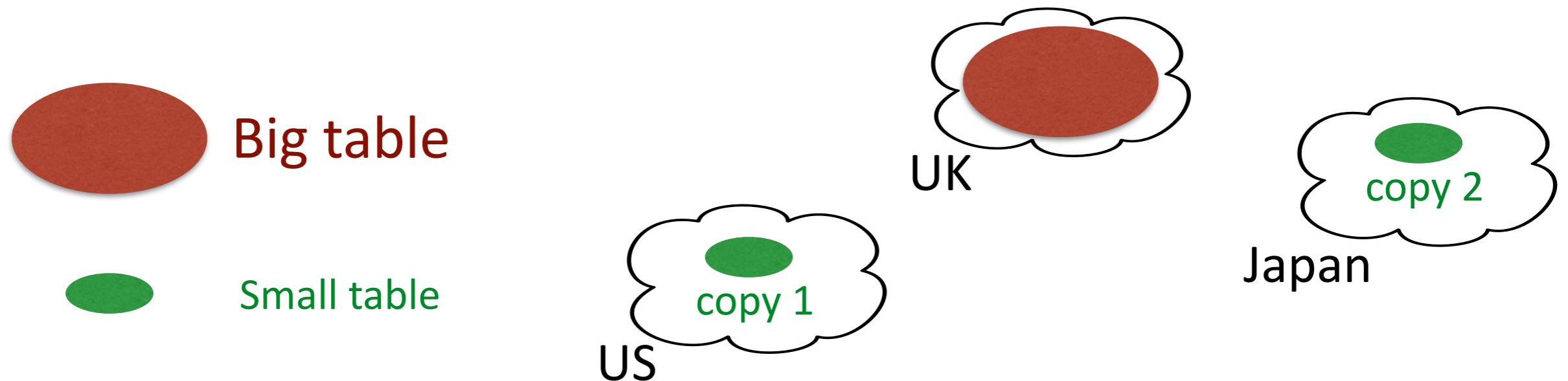
- Query plan
- Site selection (task scheduling)
- Data replication
 - Replicate data for performance and/or fault-tolerance

to minimize data transfer cost

Challenge: optimization search space is exponentially large

Approach: simplify search space

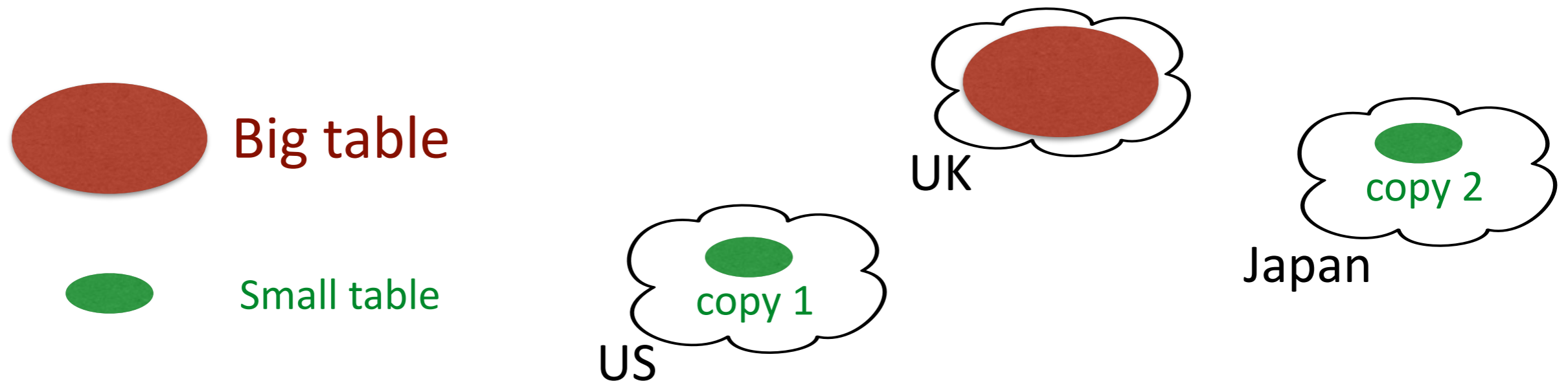
Simplification



Computation: copy both tables to one DC, then join them

Decision 1: do we copy the big table or the small table?

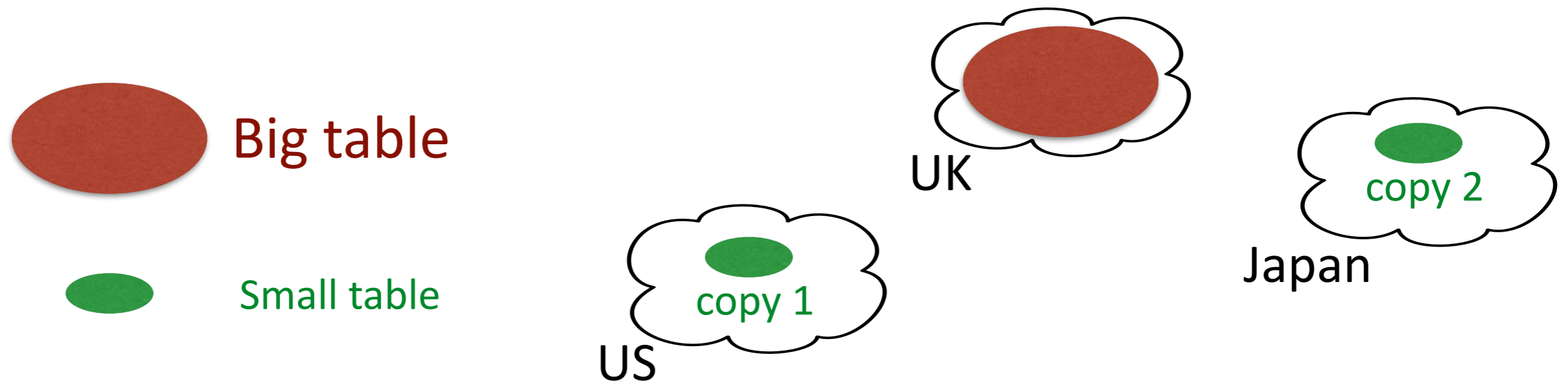
Simplification



Computation: copy both tables to one DC, then join them

Decision 1: do we copy the big table or the **small table**?

Simplification



Computation: copy both tables to one DC, then join them

Decision 1: do we copy the big table or the **small table**?

Decision 2: which copy of the small table do we use?

2. SQL-aware workload planning

Had two kinds of decisions to make:

1. **Logical plan**

- Do we copy the big table or the small table?

2. **Physical plan**

- Which copy of the small table do we use?

2. SQL-aware workload planning

Had two kinds of decisions to make:

1. **Logical plan**

- Do we copy the big table or the small table?
- Choice was clear, strategies were orders of magnitude apart

2. **Physical plan**

- Which copy of the small table do we use?
- Choice wasn't as obvious, had to know precise costs

Simplification: Two-phase approach

1. Logical plan

- Choose based on simple statistics on each table

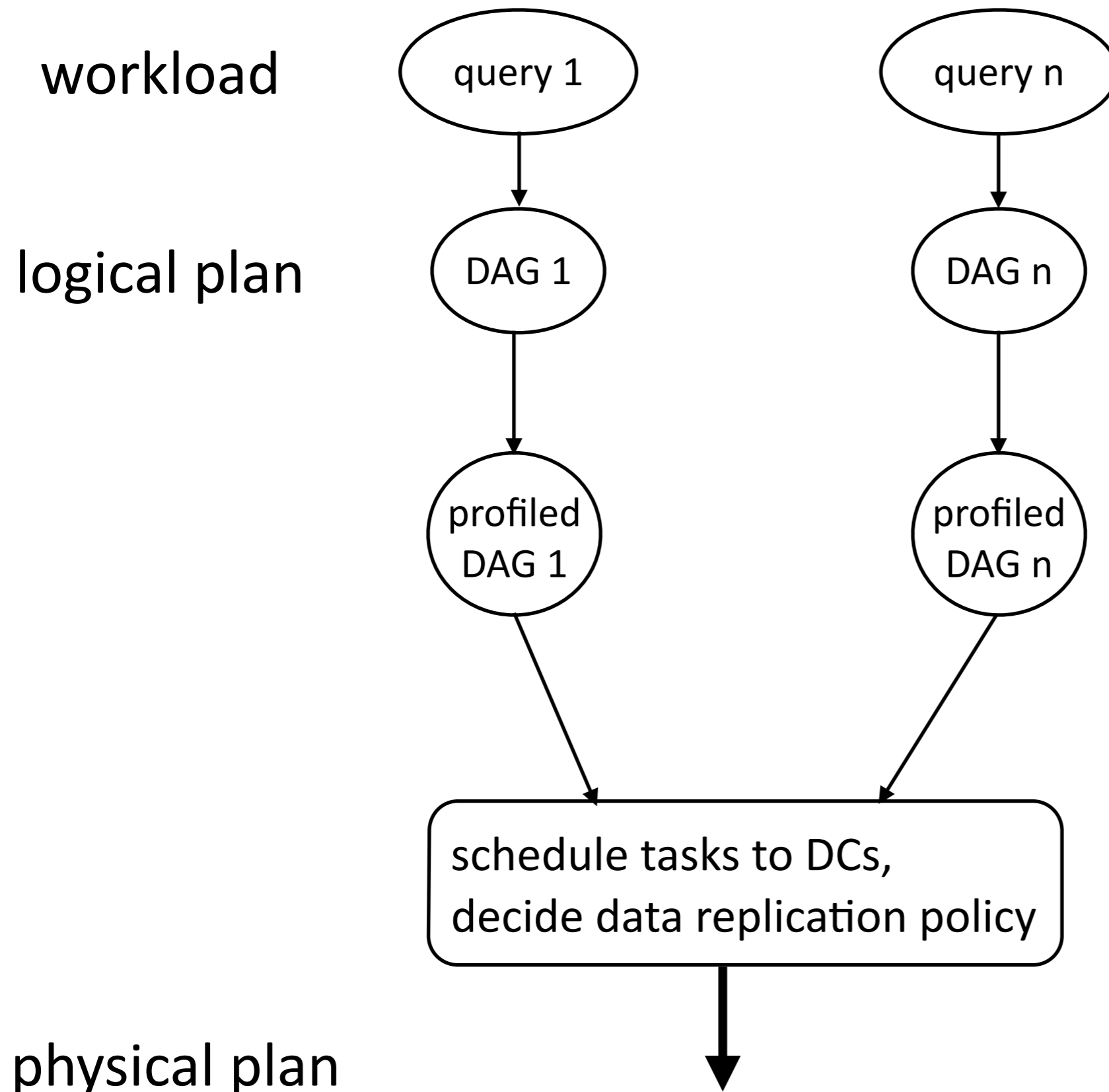
2. Physical plan

- Profile logical plan, collecting precise measurements
- Use to optimize physical plan

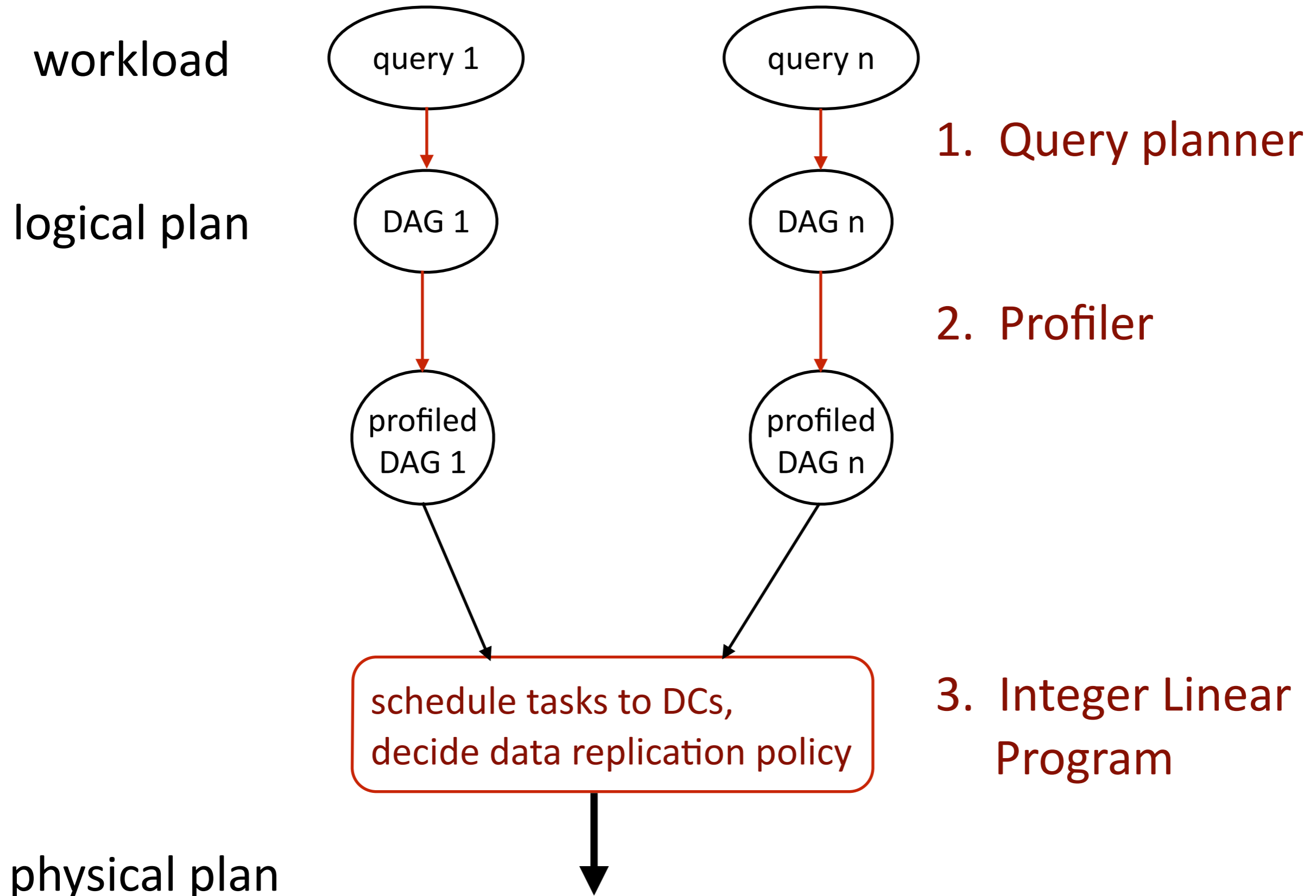
Key insight

- “Logical” choices: simple statistics usually suffice
- “Physical” choices: need more careful cost estimates
- Only an empirical insight
 - But worked well in all our experimental workloads

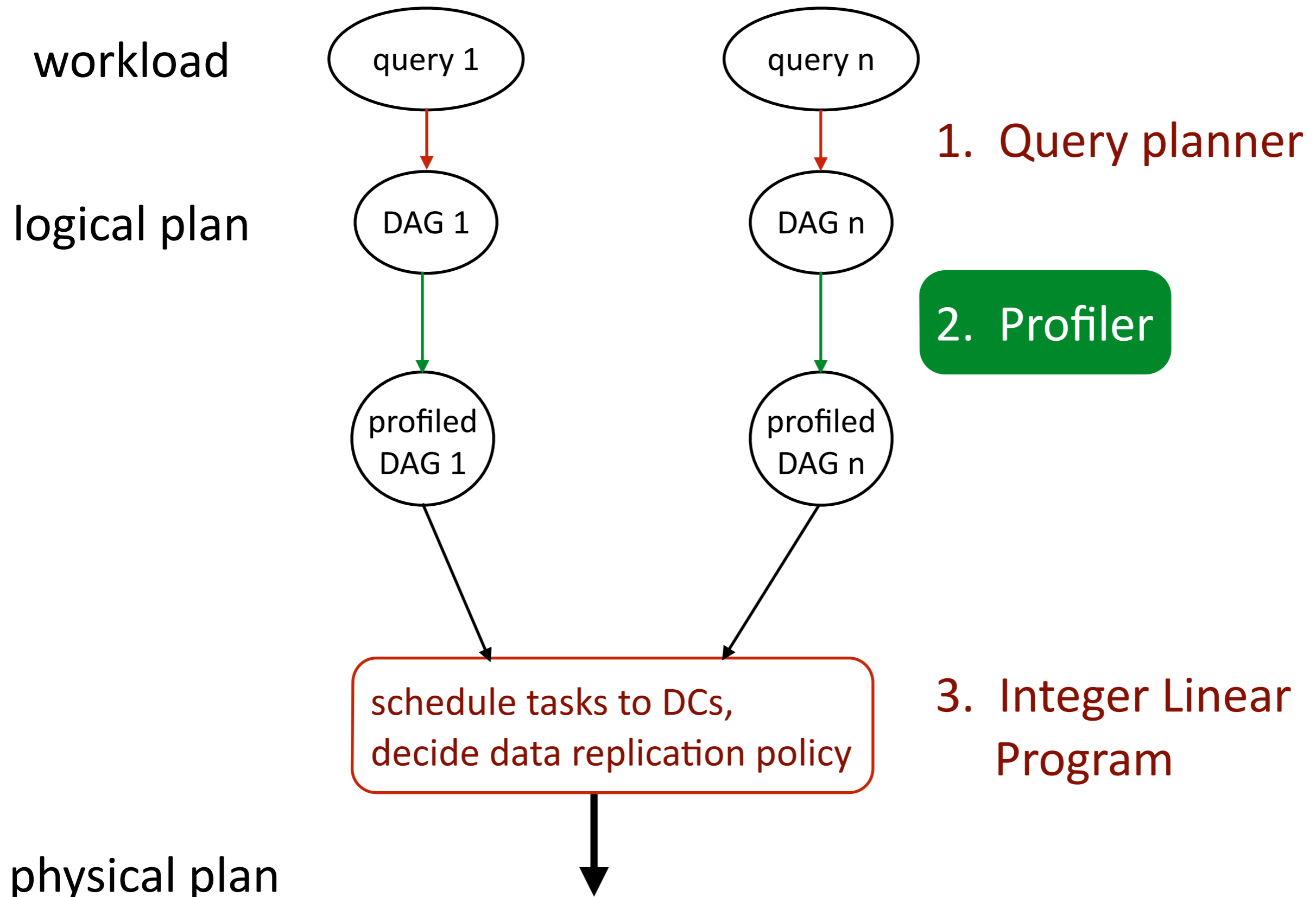
2. SQL-aware workload planning



2. SQL-aware workload planning



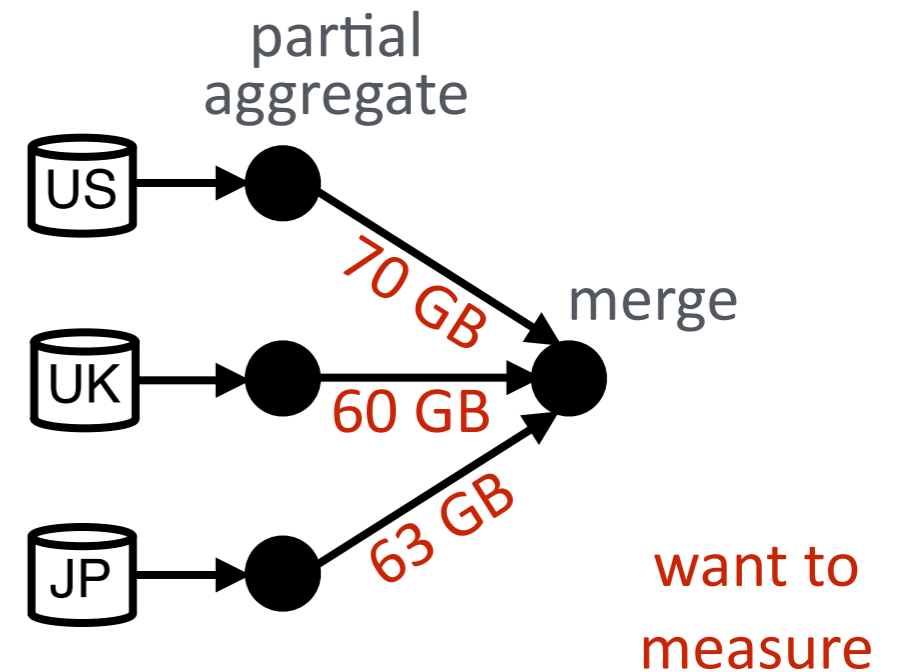
2. SQL-aware workload planning



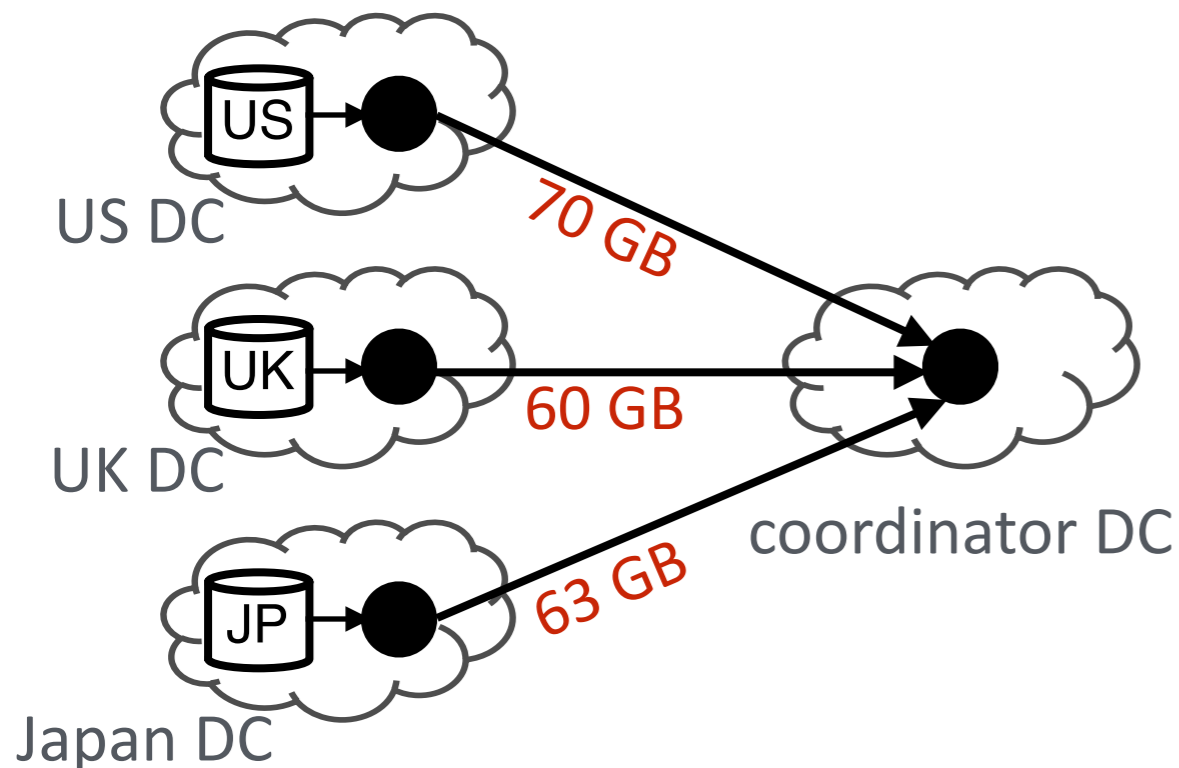
2. SQL-aware workload planning

Profiling task graphs

```
SELECT city,  
       SUM(orderValue)  
FROM sales  
WHERE category = 'Electronics'  
GROUP BY city
```



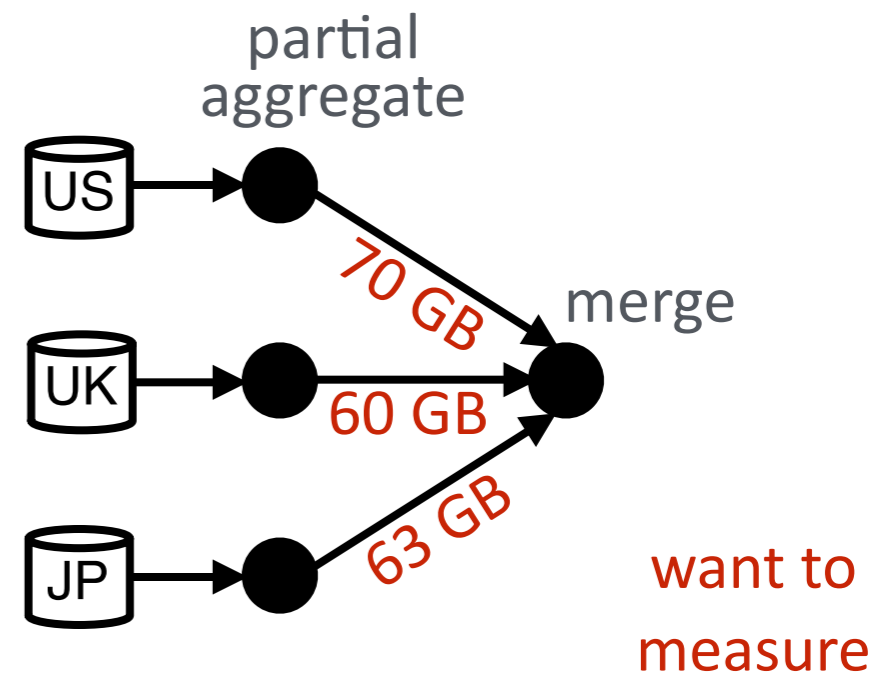
Distributed deployment:



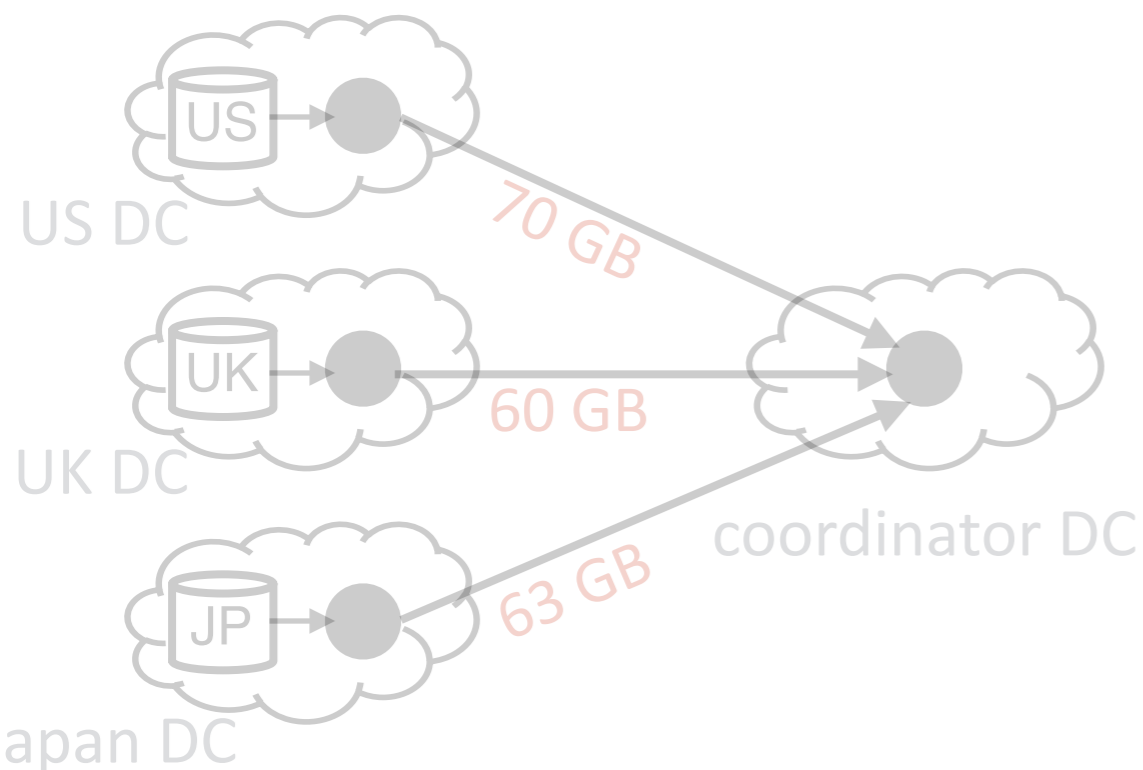
2. SQL-aware workload planning

Profiling task graphs

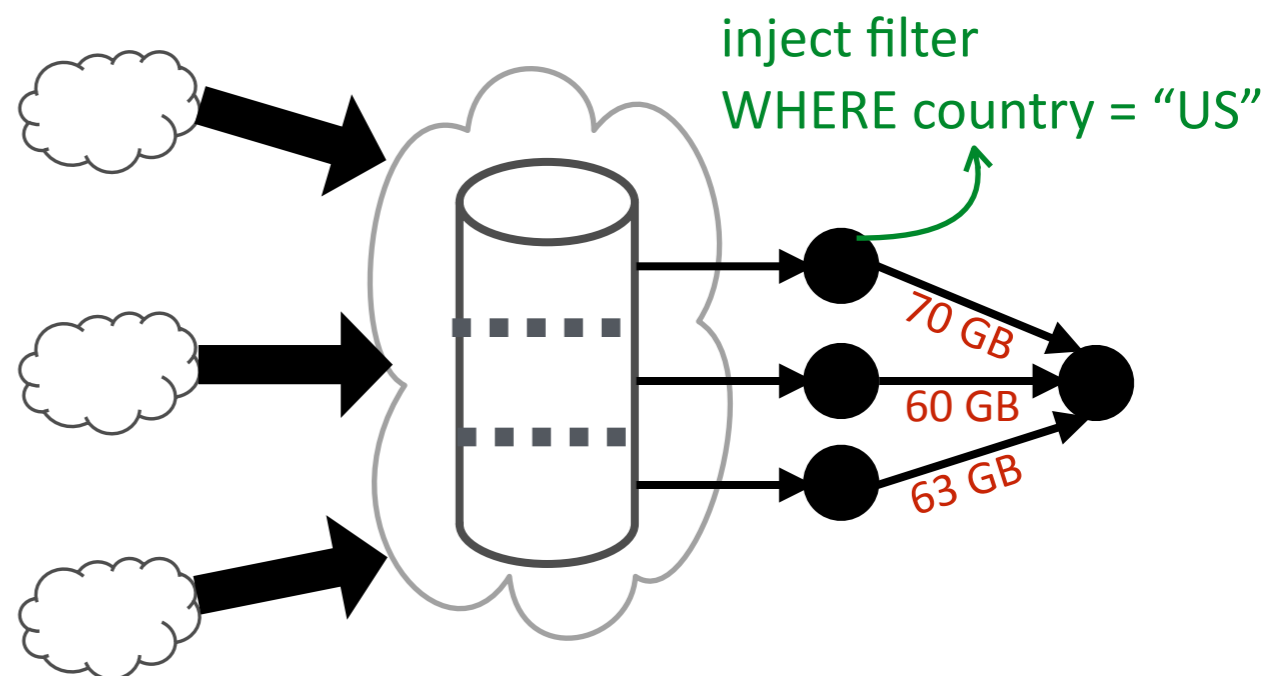
```
SELECT city,  
       SUM(orderValue)  
FROM sales  
WHERE category = 'Electronics'  
GROUP BY city
```



Distributed deployment:



Centralized deployment:



Profiling task graphs

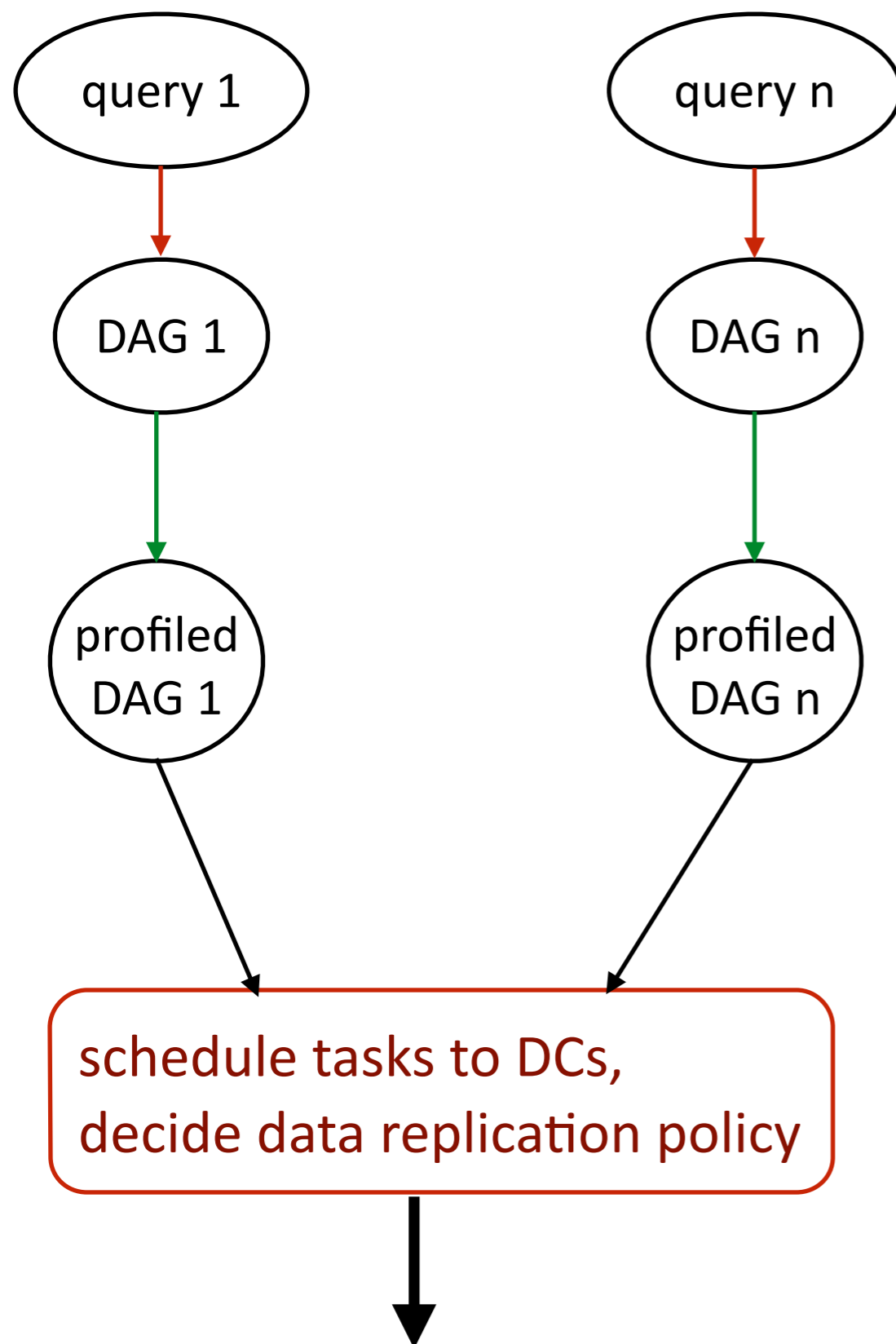
Pseudo-distributed execution

Rewrite query DAGs to simulate alternate configurations

Fully general what-if analysis. Use cases:

- Bootstrap: centralized -> distributed
- Test alternate data replication strategies
- Simulate adding/removing data centers

2. SQL-aware workload planning



1. Query Planner

2. Profiler

3. Integer Linear Program

see
paper

Optimizations

3. Function-specific

2. SQL-aware

1. Runtime

Optimizations

3. Function-specific

2. SQL-aware

1. Runtime

3. Function-specific optimizations

Past work: large number of distributed algorithms
targeting specific problems

Support via extensible user-defined function interface

- Allows registering multiple implementations
- Optimizer will automatically choose best, based on profiling

As examples, implemented

- Top-k ^[1]
- Approximate count-distinct ^[2]

[1] “Efficient top-k query computation in distributed networks”
P. Cao, Z. Wang, PODC 2004

[2] “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm”
P. Flajolet, E. Fusy, O. Gandouet, F. Meunier, AOFA 2007

EVALUATION

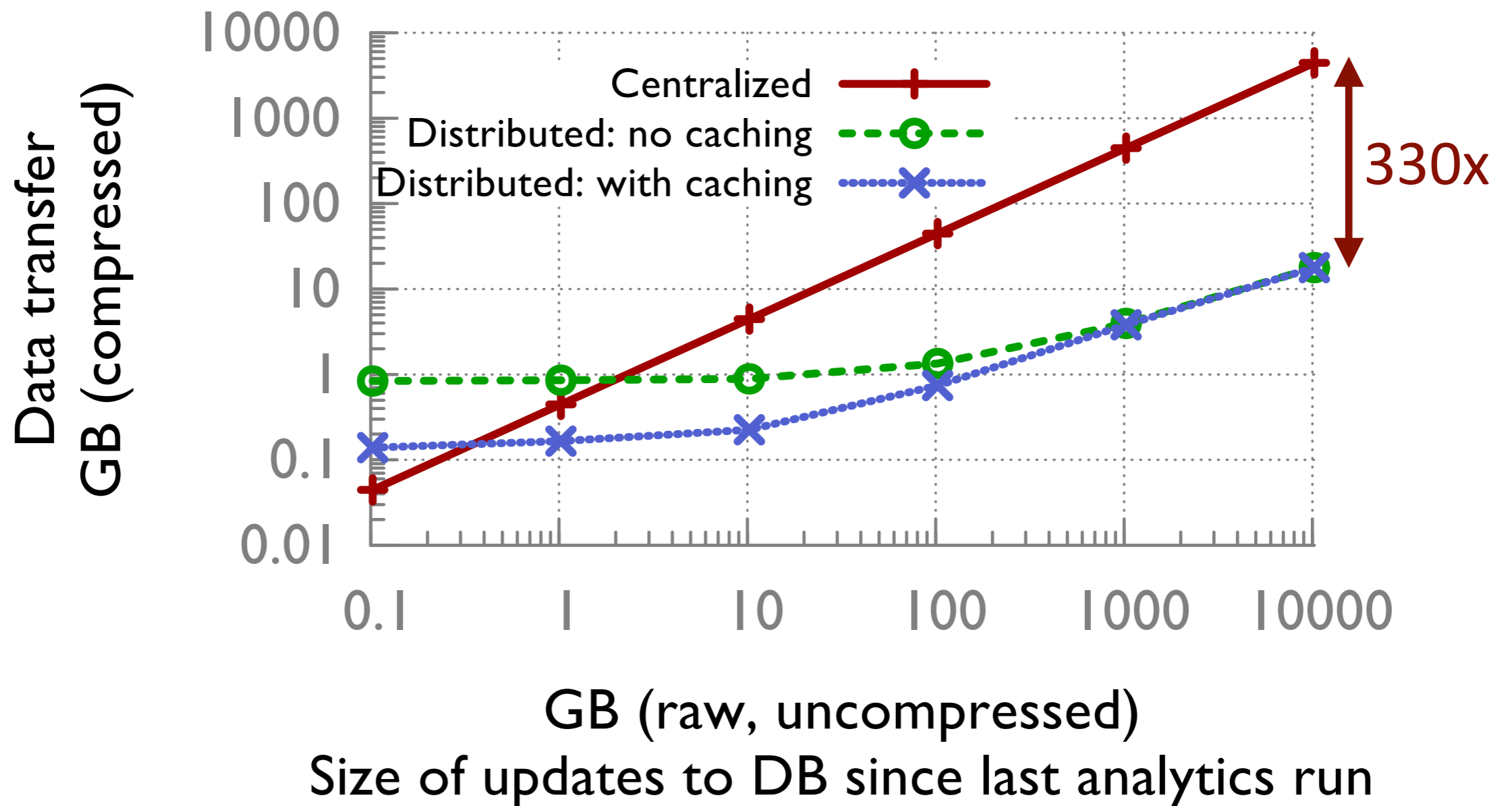
Implemented Hadoop-stack prototype

- Prototype multi-DC replacement for Apache Hive

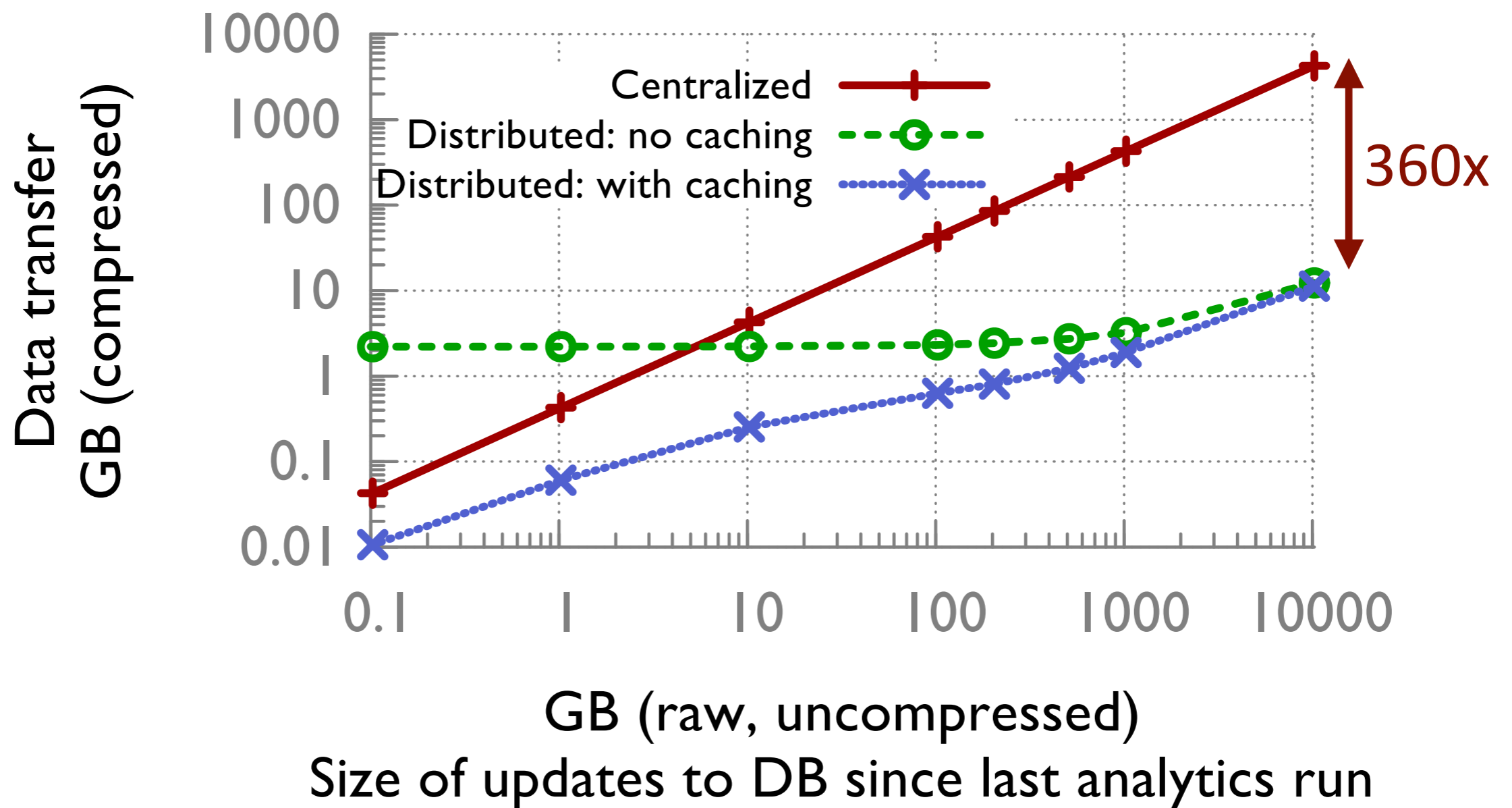
Experiments up to 10s of TBs scale

- Real Microsoft production workload
- Several synthetic benchmarks:
 - TPC-CH
 - BigBench-SQL
 - Berkeley Big-Data
 - YCSB

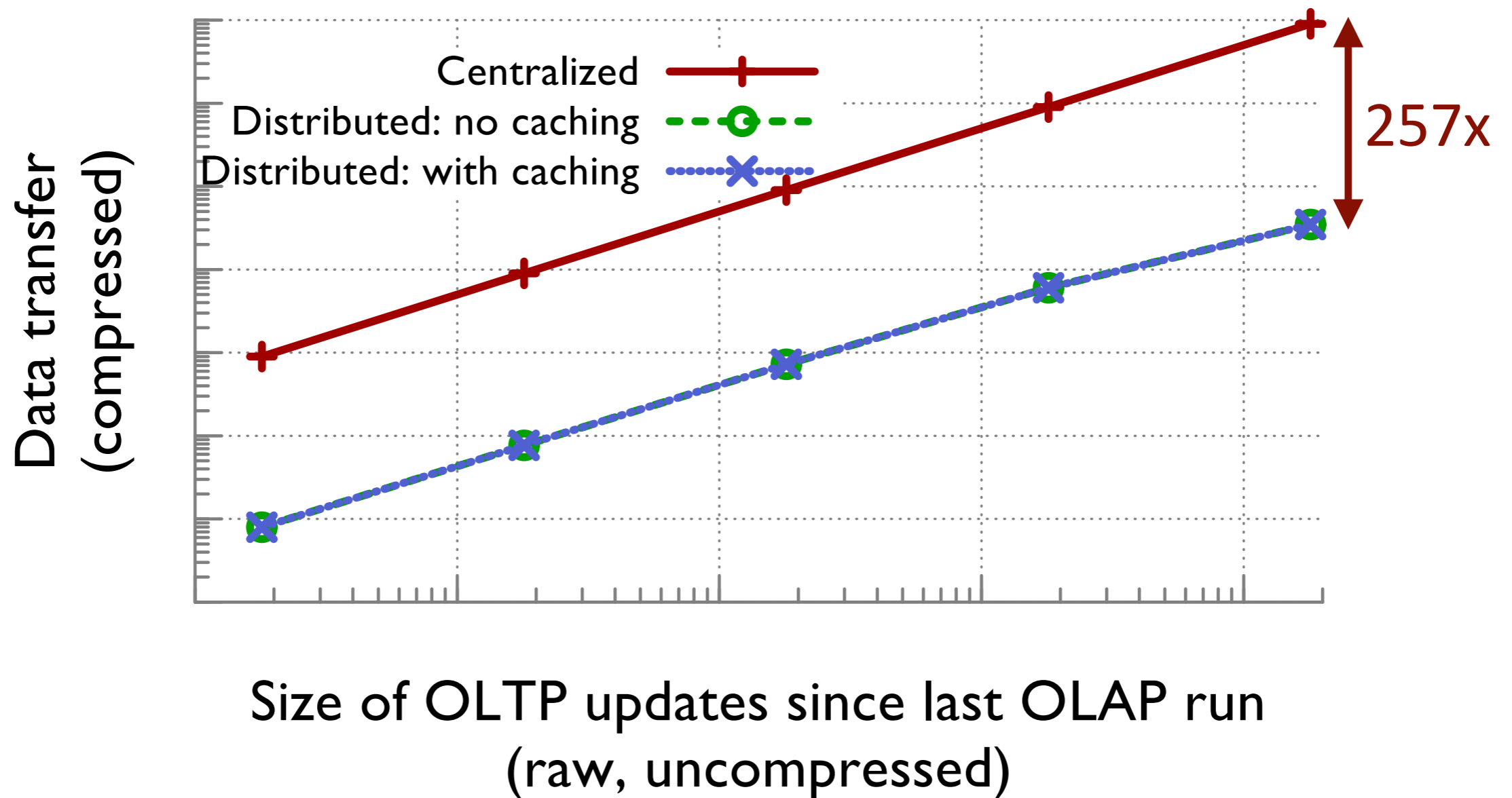
BigBench-SQL



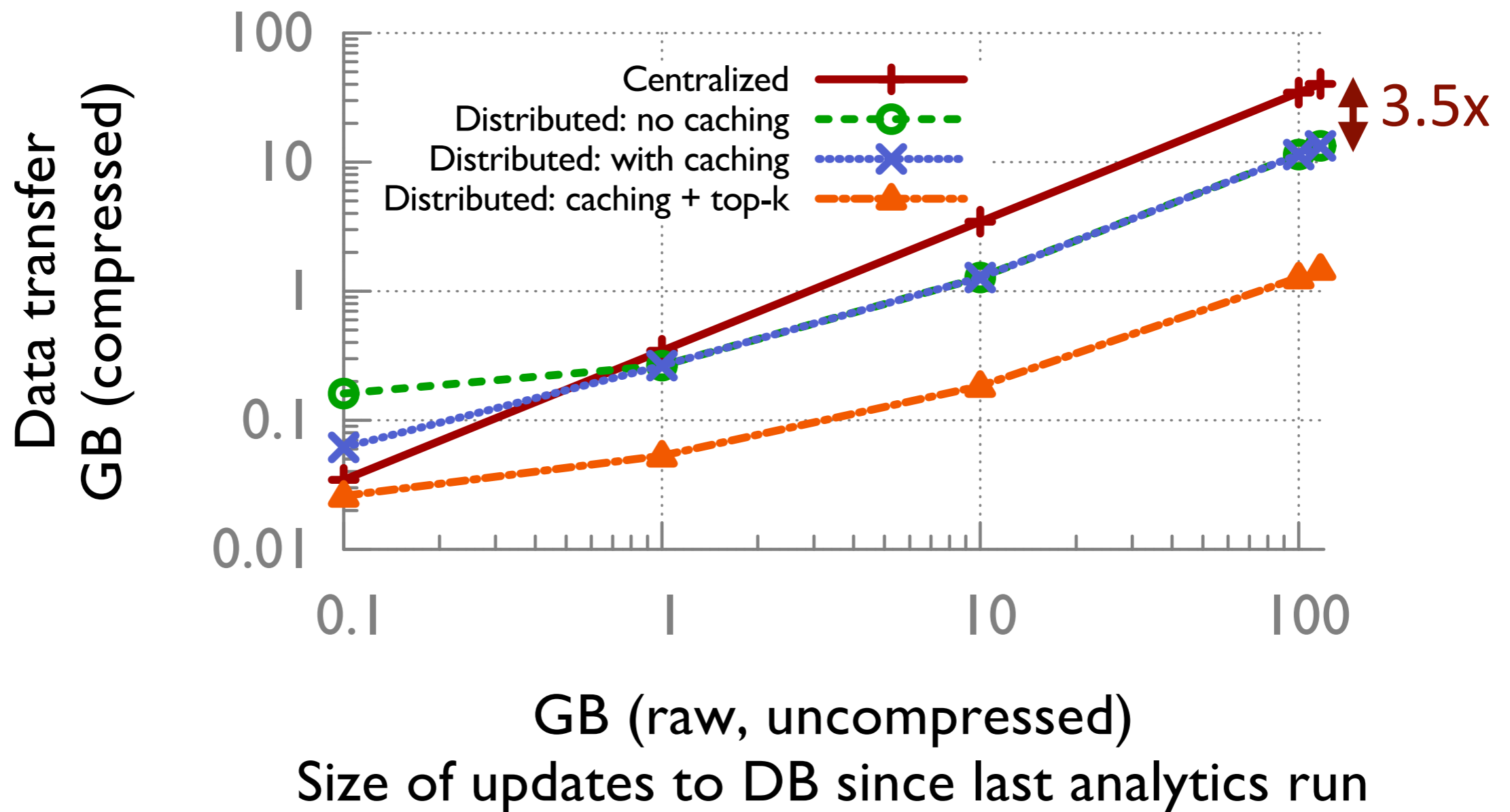
TPC-CH



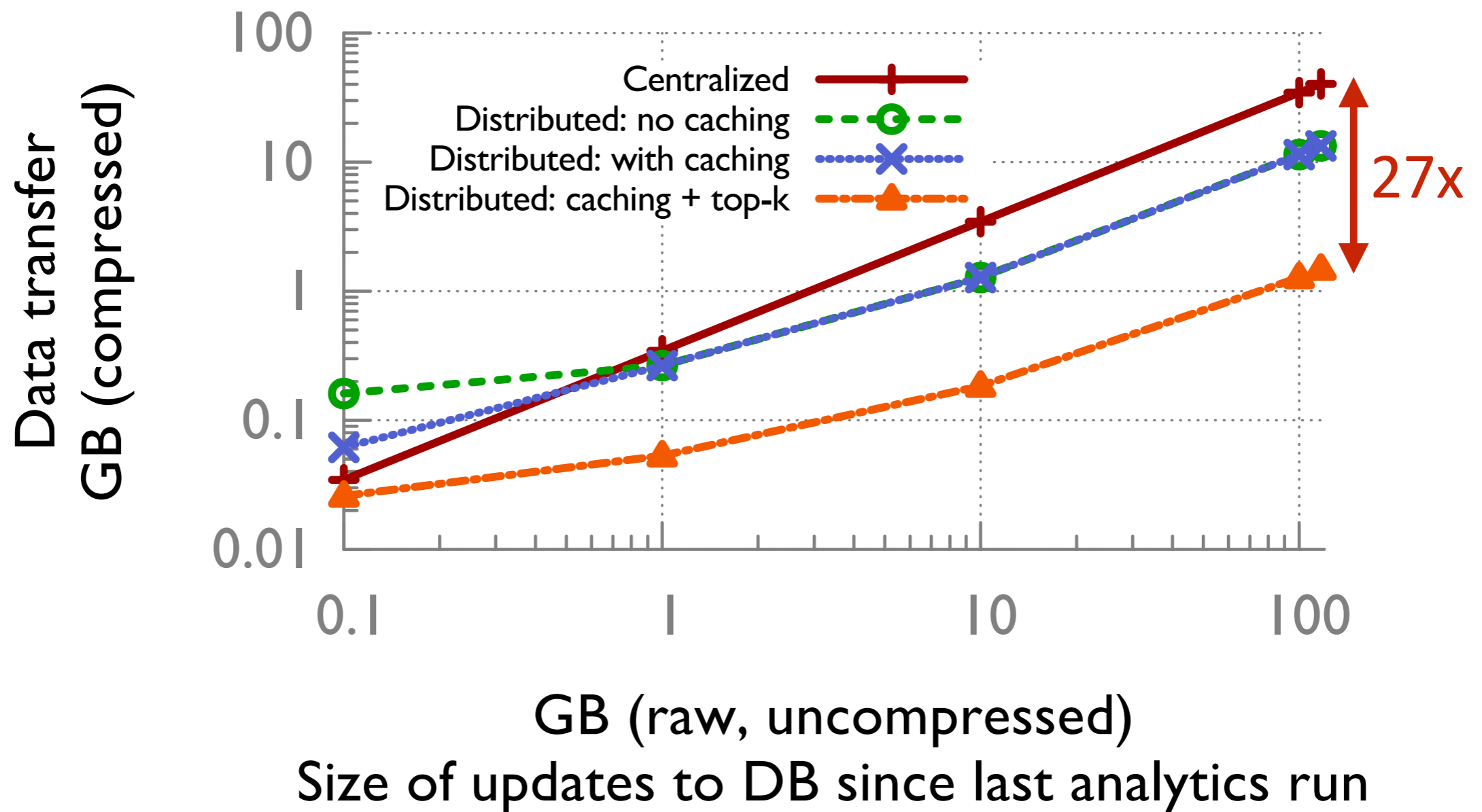
Microsoft production workload



Berkeley Big-Data



Berkeley Big-Data



BEYOND SQL

Beyond SQL: DAG workflows

Computational model: directed acyclic task graphs,
each node = arbitrary computation

Significantly more challenging setting

Initial results encouraging

- Same level of improvement as SQL

More details: [\[CIDR 2015\]](#)

RELATED WORK

Distributed and parallel databases

Single-DC frameworks (Hadoop/Spark/...)

Data warehouses

Scientific workflow systems

Sensor networks

Stream processing systems (e.g. JetStream)

...

Key characteristics

1. Support full relational model at 100s TBs/day scale
2. No control over data partitioning
3. Focus on cross-DC bandwidth
4. Unique constraints
 - Heterogeneous bandwidth costs/capacities
 - Sovereignty
5. Assumption of ~stable recurring workload
 - Enables highly tuned optimization

SUMMARY

Centralized analytics becoming unsustainable

Geo-distributed analytics: SQL and DAG workflows

Several novel techniques

- Redundancy elimination via caching
- Pseudo-distributed measurement
- [SQL query planner + ILP] optimizer

Up to **360x** less bandwidth on real & synthetic workloads

THANK YOU!

SUMMARY

Centralized analytics becoming unsustainable

Geo-distributed analytics: SQL and DAG workflows

Several novel techniques

- Redundancy elimination via caching
- Pseudo-distributed measurement
- [SQL query planner + ILP] optimizer

Up to **360x** less bandwidth on real & synthetic workloads

BACKUP SLIDES

Caching and view selection

Consider `SELECT val - avg(val) FROM table`

Cutpoint selection problem: do we cache

- Base [val], or
- Results after average has been subtracted

Akin to view selection problem in SQL databases

Current implementation makes wrong choice

Sovereignty: Partial support

Our system respects data-at-rest regulations
(e.g. German data should not leave Germany)

But we allow arbitrary queries on the data

Limitation: we don't differentiate between

- Acceptable queries, e.g.
 “what's the total revenue from each city”
- Problematic queries, e.g.
 SELECT * FROM Germany

Sovereignty: Partial support

Solution: either

- Legally vet the core workload of queries
- Use differential privacy mechanism

Open problem

Past work