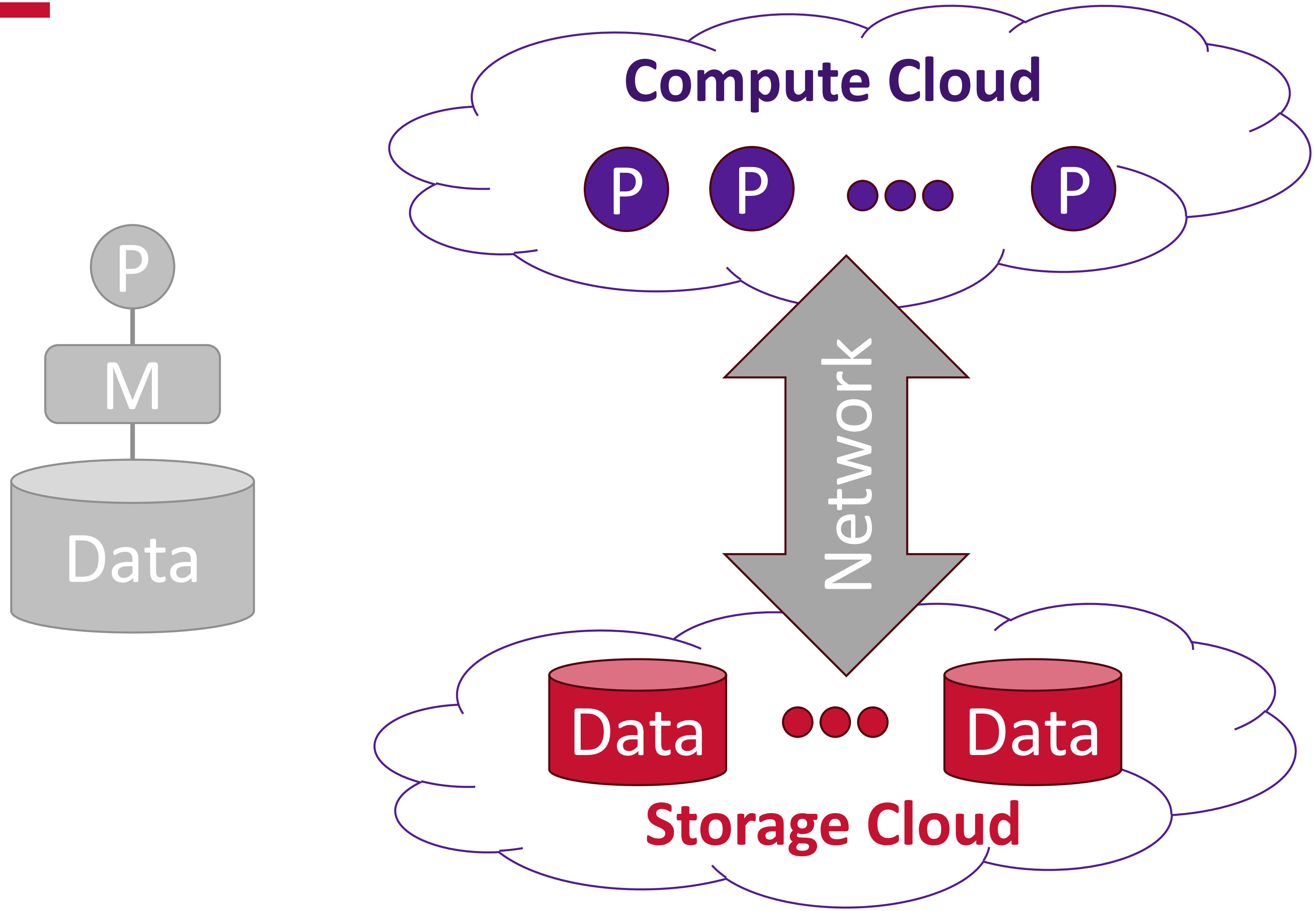
# Data analytics in a disaggregated world

Jignesh M. Patel · jignesh@cmu.edu · jigneshpatel.org

# Key transformation for data platforms




Data is stored in open formats like Parquet.

# The new world for data platforms

Data not under the direct control of the platform.

The platform has no pre-built statistics on the data being queried.

Pay-as-you-go pricing model.

Fierce competition for performance.

Storage is even further away from compute.

And there is a storage mesh not a storage hierarchy.

# Key challenges: Efficiency

**System**

- Build efficient query processing mechanisms that don't rely on pre-built statistics.

**Human**

- Search/query the data lake using natural language.

# Equijoin Query Optimization

## SSB Query 4.3

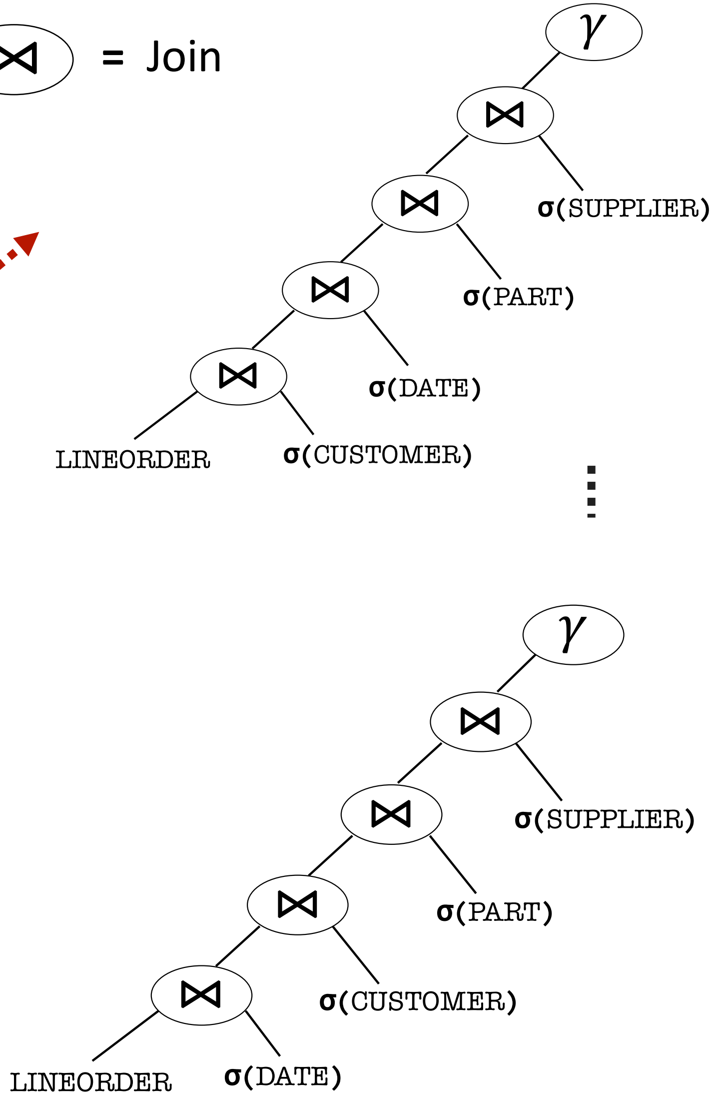
```
SELECT d_year, s_city, p_brand1,
       SUM(lo_revenue - lo_supplycost)
FROM date, customer, supplier,
     part, lineorder
WHERE lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_partkey = p_partkey
  AND lo_orderdate = d_datekey
  AND c_region = 'AMERICA'
  AND s_nation = 'UNITED STATES'
  AND (d_year = 1997
       OR d_year = 1998)
  AND p_category = 'MFGR#14'
GROUP BY d_year, s_city, p_brand1
ORDER BY d_year, s_city, p_brand1;
```

**Query Optimizer**

- Cardinality Estimation
- Plan Space Enumeration
- Cost Estimation

$\gamma$ = Aggregation

$\bowtie$ = Join



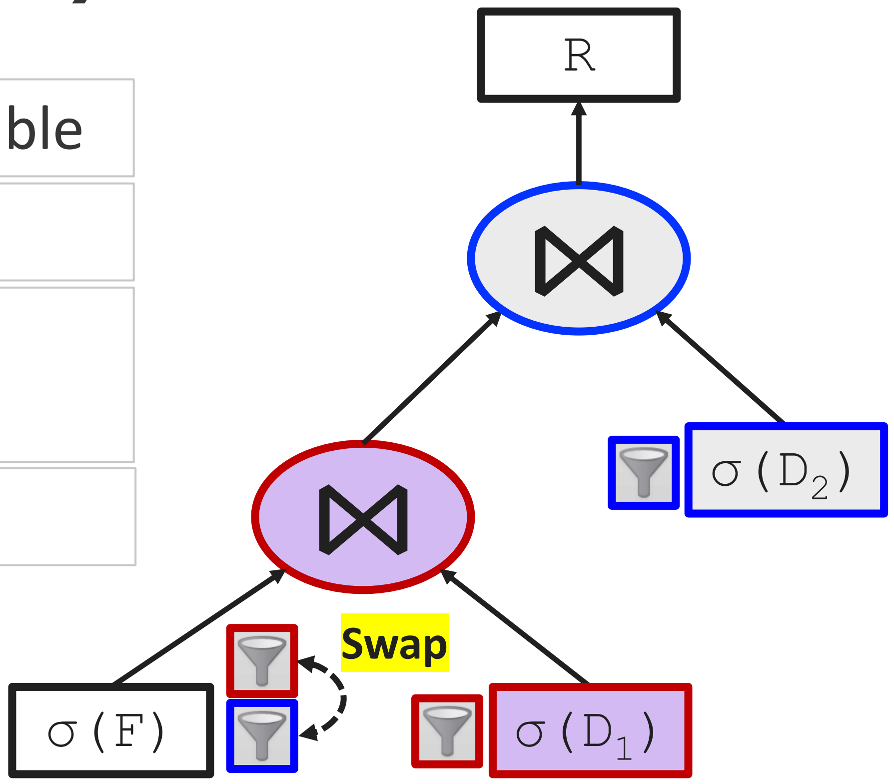

**30+ year old problem**: Cardinality estimation errors grow exponentially over successive joins.

Y. E. Ioannidis and S. Christodoulakis. *On the propagation of errors in the size of join results.* SIGMOD, 1991.

Zhu, Potti, Saurabh, Patel: Looking Ahead Makes Query Plans Robust. **VLDB 2017**

# Lookahead Information Passing (LIP)

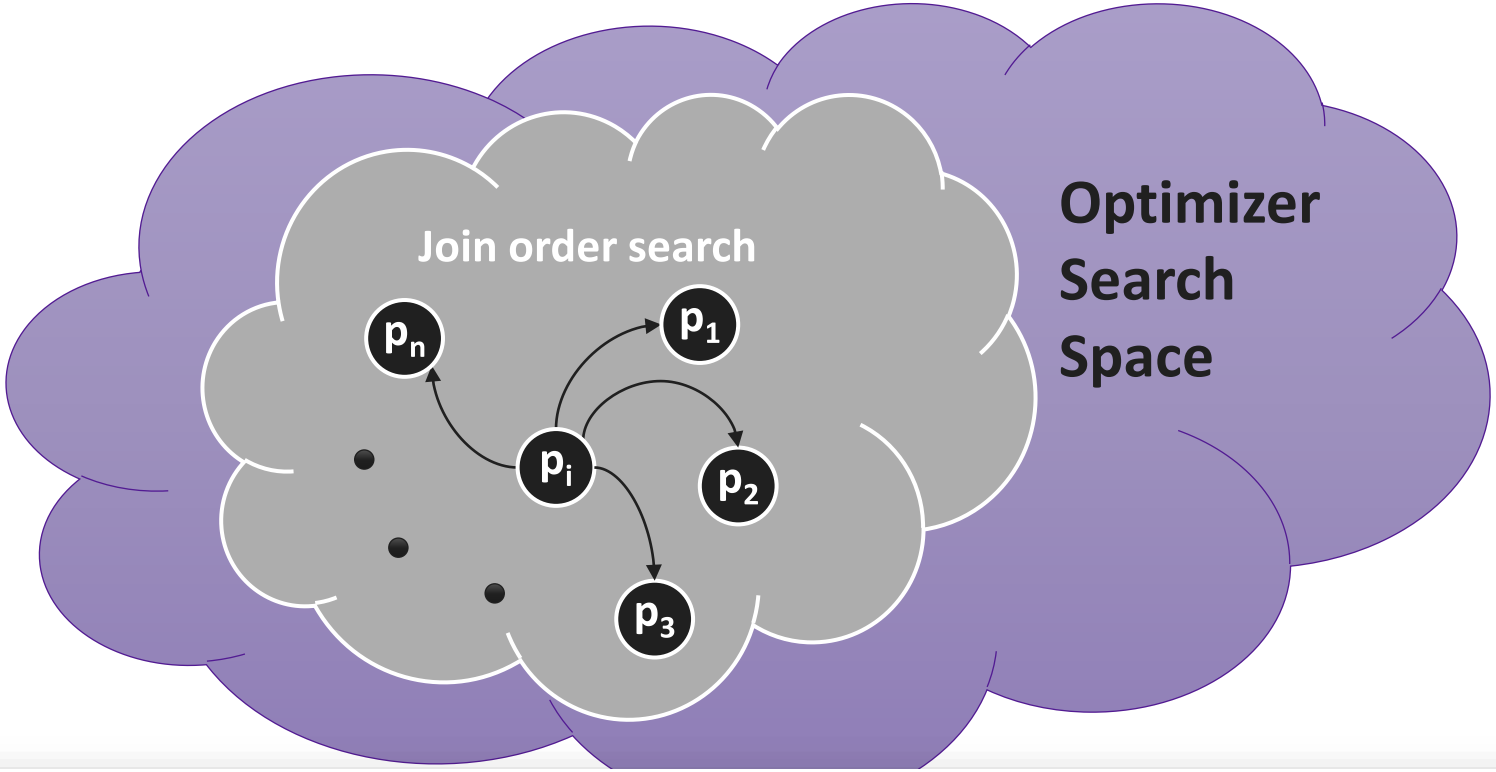
1. Build a filter with the hash table for each dimension table
2. Pass all filters to the fact table scan operator
3. **Adapt** the filter order on a sample using a multi-arm bandit algorithm
4. Apply the filter before probing the hash table




Observation: Adaptive filtering converges to the "optimal" join order

Implication: No need to optimize a linear equijoin subtree

# No need to optimize linear equijoin subtrees




**Looking Ahead Makes Query Plans Robust**

Making the Initial Case with In-Memory Star Schema Data Warehouse Workloads

Jianqiao Zhu Navneet Potti Saket Saurabh Jignesh M. Patel
Computer Sciences Department
University of Wisconsin–Madison
{jianqiao, nav, saket, jignesh}@cs.wisc.edu

**ABSTRACT**

Query optimizers and query execution engines cooperate to deliver high performance on complex analytic queries. Typically, the optimizer searches through the plan space and sends a selected plan to the execution engine. However, optimizers may at times miss the optimal plan, with sometimes disastrous impact on performance. In this paper, we develop the notion of robustness of a query evaluation strategy with respect to a space of query plans. We also propose a novel query execution strategy called *Lookahead Information Passing (LIP)* that is robust with respect to the space of (fully pipeline-able) left-deep query plan trees for in-memory star schema data warehouses. LIP ensures that execution times for the best and the worst case plans are far closer than without LIP. In fact, under certain assumptions of independent and uniform distributions, any plan in that space is theoretically guaranteed to execute in near-optimal time. LIP ensures that the execution time for every plan in the space is nearly-optimal. In this paper, we also evaluate these claims using workloads that include skew and correlation. With LIP we make an initial foray into a novel way of thinking about robustness from the perspective of query evaluation, where we develop strategies (like LIP) that collapse plan sub-spaces in the overall global plan space.

**1. INTRODUCTION**

Relational database management systems (RDBMSs) have a unique internal organization where query execution can be viewed as a composition of basic relational algebraic (RA) operations. This underlying framework allows RDBMSs to navigate the space of equivalent RA compositions to find the most efficient execution plan. This ability to optimize query plans is crucial to the RDBMSs' ability to execute complex queries efficiently even on large databases.

Query optimization, however, is a complex task. Decades of research in this area have yielded a plethora of techniques for plan enumeration, cardinality and cost estimation, and dynamic query optimization. Despite these remarkable advancements, it is well known [17, 23] that query optimizers still falter in some cases, producing query plans that have disastrously worse performance than optimal.

Rather than directly improving the capability of query optimizers, in this work, we take an approach that is complementary to most prior work in query optimization. The question we seek to address is: *Can we develop query execution techniques that dramatically reduce the impact of a poor choice of a query plan?* Thus, the big picture view of our approach is to focus on developing efficient *query evaluation techniques* that increase the *robustness* of query plans, by mitigating issues related to bad plan selection within a subspace of plans. We further limit our scope to increasing the robustness of plans to errors in join order selection.

*Lookahead Information Passing* (LIP), the query evaluation strategy that we propose in this work, is targeted at the common scenario of star schema data warehouses. In such workloads, a natural space of "good" plans for a query optimizer is that of fully pipeline-able left-deep join trees. For instance, consider Query 4.3 in the Star Schema Benchmark, shown in Figure 2. Figures 2a and 2b show two of the 24 possible left-deep query plans for this query, resulting from all permutations of the 4 dimension tables in the query.

If the optimizer selects a poor join order for such a query, the intermediate join results will be needlessly large, incurring additional processing time for extraneous tuples. One approach to reducing the impact of bad plan selection, therefore, is to efficiently pre-filter such extraneous tuples. This idea underlies our proposed LIP strategy.

In essence, the LIP strategy consists of two components. First, we pass succinct filter data structures (such as Bloom filters) from the "outer" (dimension) relations in all the joins to the "inner" (fact) relation. Thus, we can approximately pre-filter the fact table before performing the join opera-



**Figure 1:** All 24 possible left-deep query plans for SSB Query 4.3 in increasing order of execution time.


*Proceedings of the VLDB Endowment*, Vol. 10, No. 8

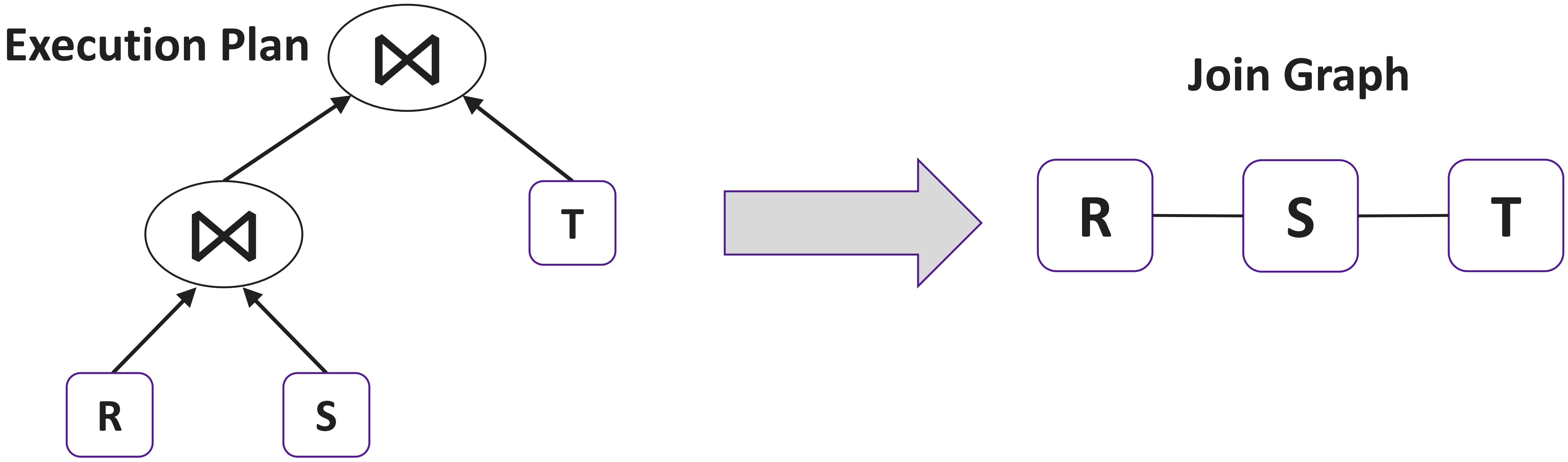

**Definition 1** *$\Theta$-Robustness: An evaluation strategy $\mathcal{E}$ is said to be $\Theta$-robust with respect to a plan space $\mathcal{P}$ if the maximum deviation in performance of any plan in $\mathcal{P}$ (including the worst plan $\mathcal{E}_w$) from the best one $\mathcal{E}_b$, normalized by the fact table cardinality and spread of selectivities in a query, is at most $\Theta$.*

$$\frac{T(\mathcal{E}_w) - T(\mathcal{E}_b)}{(\sigma_{max} - \sigma_{min})|F|} \leq \Theta, \qquad \sigma_{max} \neq \sigma_{min} \qquad \boxed{\Theta = \frac{1}{2}\frac{\sigma_1\sigma_2\ldots\sigma_n}{\sigma_{\min}\sigma_{\max}}\epsilon n(n+1)}$$

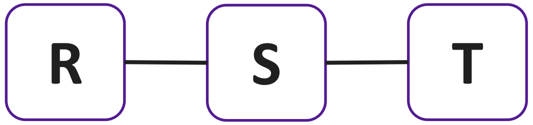
# Going beyond star schema join trees

## The Yannakakis Algorithm [VLDB'81]

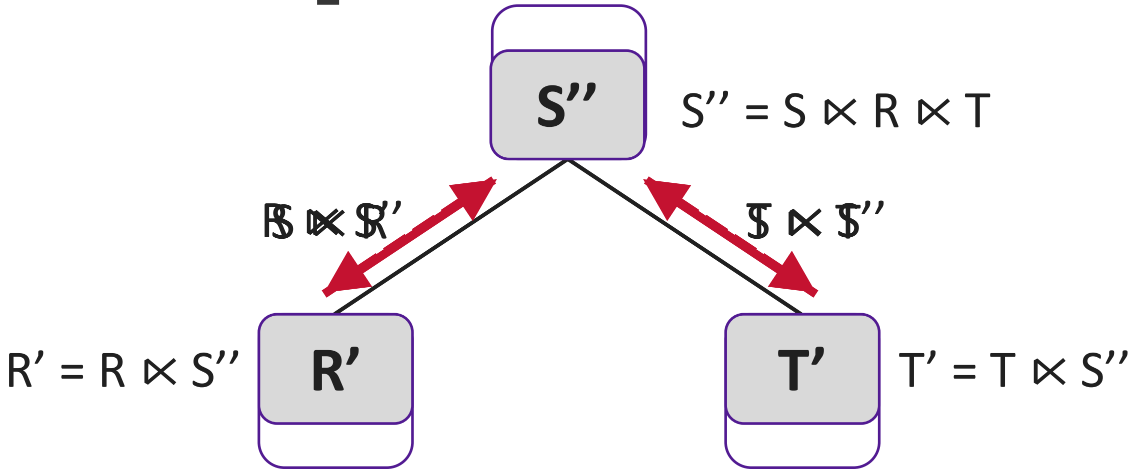
For any acyclic conjunctive query, the query can be evaluated in polynomial time w.r.t. the size of the database!



# The Yannakakis Algorithm [VLDB'81]

- Join Graph: 

- Pick up the join graph at some node:




- Upward pass:
  - S' = S ⋉ R (filter S to only include tuples that match with R).
  - S'' = S' ⋉ T (further filter S to only include tuples that match with T).
- Downward pass:
  - R' = R ⋉ S'' (filter R).
  - T' = T ⋉ S'' (filter T).
- Join phase: 

# Open research questions

How far can we push 1-pass algorithms to cover more general query shapes?

If two passes are needed, what is the impact of the "schedule" on making these two passes when the query graph is more complex?

Are there better summary structures than bloom filters for these cascaded semijoin operations?

# Key challenges: Efficiency

**System**

- Build efficient query processing mechanisms that don't rely on pre-built statistics.

**Human**

- Search/query the data lake using natural language.

# Querying data using natural language


| *Cyclist* | *Rank* |
|---|---|
| *Alejandro (ESP)* | *1* |
| *Alexandr (RUS)* | *2* |
| … | … |

**Key challenge**: Work with messy data.

Initial Scope: Single tables, e.g. Wikipedia tables or CSV files.

## ReAcTable: Enhancing ReAct for Table Question Answering


Yunjia Zhang
University of Wisconsin-Madison
yunjia@cs.wisc.edu

Jordan Henkel
Microsoft
jordan.henkel@microsoft.com

Avrilia Floratou
Microsoft
avflor@microsoft.com

Joyce Cahoon
Microsoft
jcahoon@microsoft.com

Shaleen Deep
Microsoft
shaleen.deep@microsoft.com

Jignesh M. Patel*
Carnegie Mellon University
jignesh@cmu.edu



### ABSTRACT

Table Question Answering (TQA) presents a substantial challenge at the intersection of natural language processing and data analytics. This task involves answering natural language (NL) questions on top of tabular data, demanding proficiency in logical reasoning, understanding of data semantics, and fundamental analytical capabilities. Due to its significance, a substantial volume of research has been dedicated to exploring a wide range of strategies aimed at tackling this challenge including approaches that leverage Large Language Models (LLMs) through in-context learning or Chain-of-Thought (CoT) prompting as well as approaches that train and fine-tune custom models.

Nonetheless, a conspicuous gap exists in the research landscape, where there is limited exploration of how innovative foundational research, which integrates incremental reasoning with external tools in the context of LLMs, as exemplified by the ReAct paradigm, could potentially bring advantages to the TQA task. In this paper, we aim to fill this gap, by introducing ReAcTable (**ReAct** for **Table** Question Answering tasks), a framework inspired by the ReAct paradigm that is carefully enhanced to address the challenges uniquely appearing in TQA tasks such as interpreting complex data semantics, dealing with errors generated by inconsistent data and generating intricate data transformations. ReAcTable relies on external tools such as SQL and Python code executors, to progressively enhance the data by generating intermediate data representations, ultimately transforming it into a more accessible format for answering the user's questions with greater ease. Through extensive empirical evaluations using three popular TQA benchmarks, we demonstrate that ReAcTable achieves remarkable performance even when compared to fine-tuned approaches. In particular, it outperforms the best prior result on the WikiTQ benchmark, achieving an accuracy of 68.0% without requiring training a new model or fine-tuning.



**PVLDB Reference Format:**
Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. ReAcTable: Enhancing ReAct for Table Question Answering . PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX


*Work done while at U. Wisconsin.




Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/yunjiazhang/ReAcTable.git.


### 1 INTRODUCTION

Table question answering (TQA) [16] is a subfield of natural language processing (NLP) and information retrieval that focuses on answering natural language (NL) questions over tabular data such as Wikipedia tables, spreadsheets or relational tables. It constitutes a complex task that demands a fusion of contextual understanding, logical reasoning and analytical skills. TQA allows users without expertise in querying languages and data analytics to interact with their data using plain language and gain valuable insights. It is a vital tool that can enhance data accessibility, usability, and decision support across various domains, ultimately leading to more efficient and informed decision-making processes.

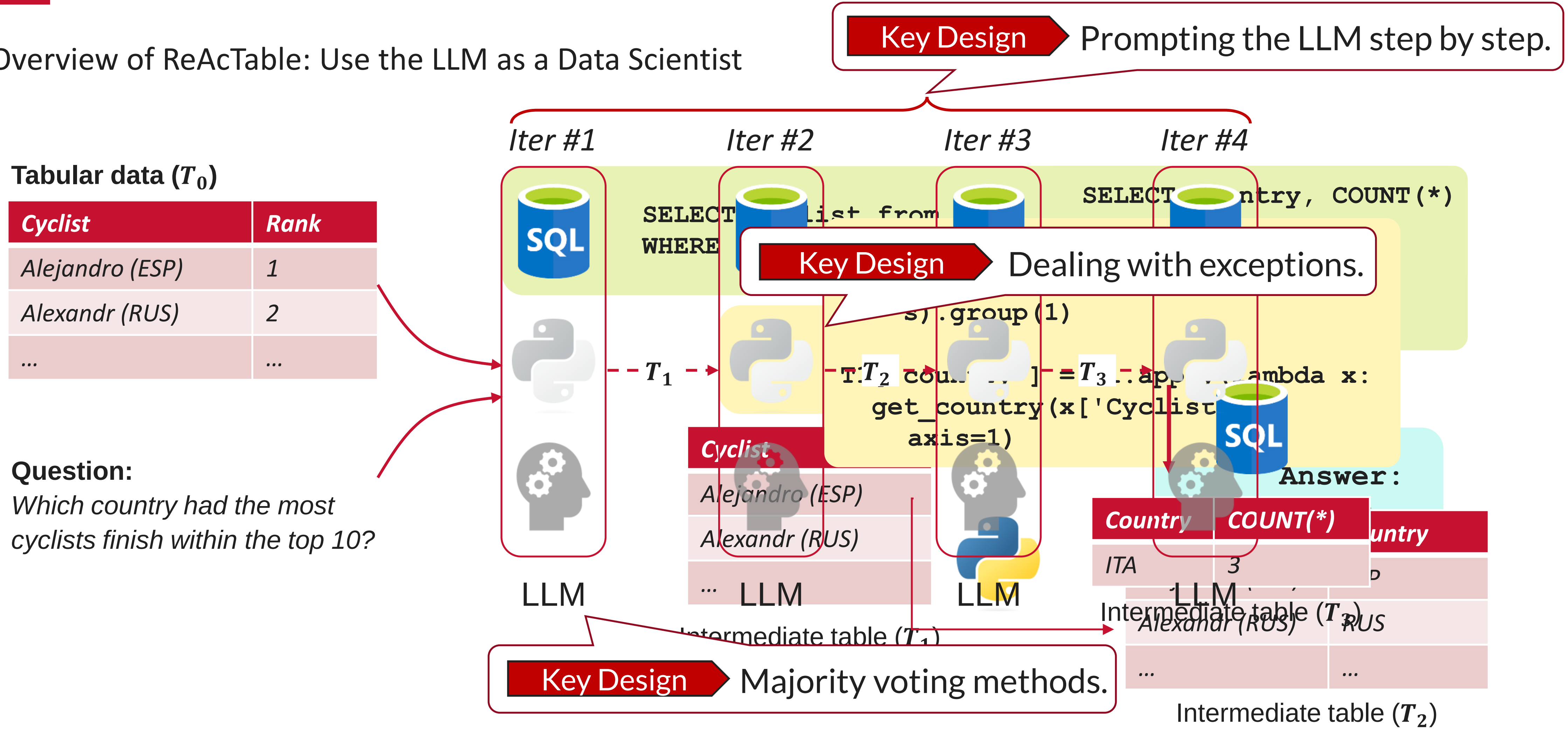
Recognizing its significance, extensive research efforts have been dedicated to devising effective strategies for TQA. These strategies can be broadly classified into two categories. In the first category, approaches such as Tapas [12], Tapex [23], Tacube [57], and OmniTab [15] involve the training or fine-tuning of specialized models tailored for the task. The second category capitalizes on recent advancements in Large Language Models (LLMs). Within this category, works like [5, 26, 50] harness LLMs to generate code capable of manipulating tabular data.

The emergence of Chain-of-Thought (CoT) prompting, which encourages a model to engage in step-by-step reasoning, has brought about a significant transformation in the utilization of Large Language Models (LLMs) for intricate multi-step tasks. Expanding the CoT ideas, the ReAct paradigm [49] has been introduced, enabling interactions between the model and external tools in an interleaved manner. This allows for greater synergy between reasoning and acting and facilitates real-time guidance and corrections during task execution. These innovative strategies aim to address the limitations of traditional few-shot prompting methods [2]. Despite the promising results demonstrated by combining reasoning with external tools, to the best of our knowledge, the ReAct paradigm has not yet been applied to the TQA task.

This paper bridges this gap by investigating how the principles behind the ReAct framework, i.e. CoT and availability of external tools, can be applied to the TQA task. Beyond the anticipated difficulty of accurately comprehending the user's natural language query, the TQA task poses a series of distinct challenges, including: (i) interpreting potentially intricate data semantics, (ii) the presence of noisy or inconsistent data, and (iii) the necessity for complex

# ReAcTable: Overview

Overview of ReAcTable: Use the LLM as a Data Scientist



# Evaluation

**Table 1: Performance of ReAcTable on WikiTQ data set.**

| Methods | Accuracy |
|---|---|
| ***Approaches require training*** | |
| Tapex | 57.5% |
| TaCube | 60.8% |
| OmniTab | 62.8% |
| Lever | **62.9%** |
| ***Approaches without training*** | |
| Binder | 61.9% |
| Dater | 65.9% |
| ReAcTable | 65.8% |
| *with s-vote* | **68.0%** |
| *with t-vote* | 66.4% |
| *with e-vote* | 67.2% |

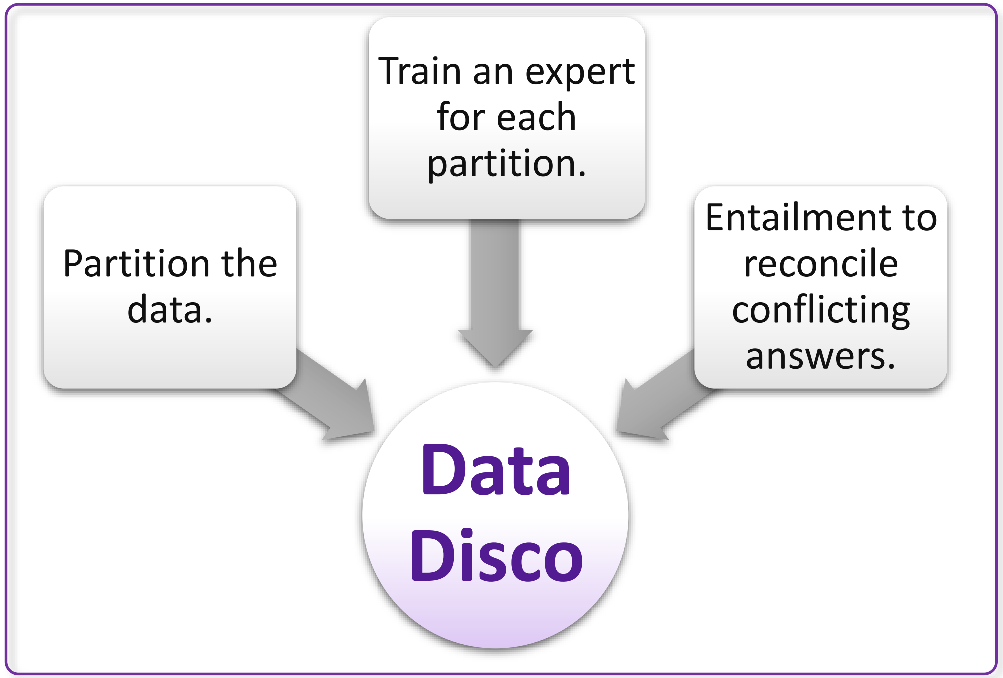
**Table 2: Performance of ReAcTable on TabFact data set.**

| Methods | Accuracy |
|---|---|
| ***Approaches require training*** | |
| TaPas | 83.9% |
| Tapex | 86.7% |
| SaMoE | 86.7% |
| PASTA | **90.8%** |
| ***Approaches without training*** | |
| Binder | 85.1% |
| Dater | 85.6% |
| ReAcTable | 83.1% |
| *with s-vote* | **86.1%** |
| *with t-vote* | 84.2% |
| *with e-vote* | 84.9% |

# Data Disco: Data discovery over data lakes

**Problem**: Data lakes often have thousands of table.

Large Data Lake




**Task**: Help the user identify which tables are relevant.

"What type of organization employs the most research staff?"

Natural Language Query

organizations

research_staff

Relevant Tables

Joint work with Leon Lu, Yunjia Zhang and Theo Rekatsinas.

# Key challenges with using LLM in data platforms

**Effectiveness**

Especially when dealing with complex data and messy data

**Efficiency**

Especially when the calls are in the “inner loop.”

**Repeatability**

Make the overall system deterministic or as close to it as possible.

NL Task → DSL → Code