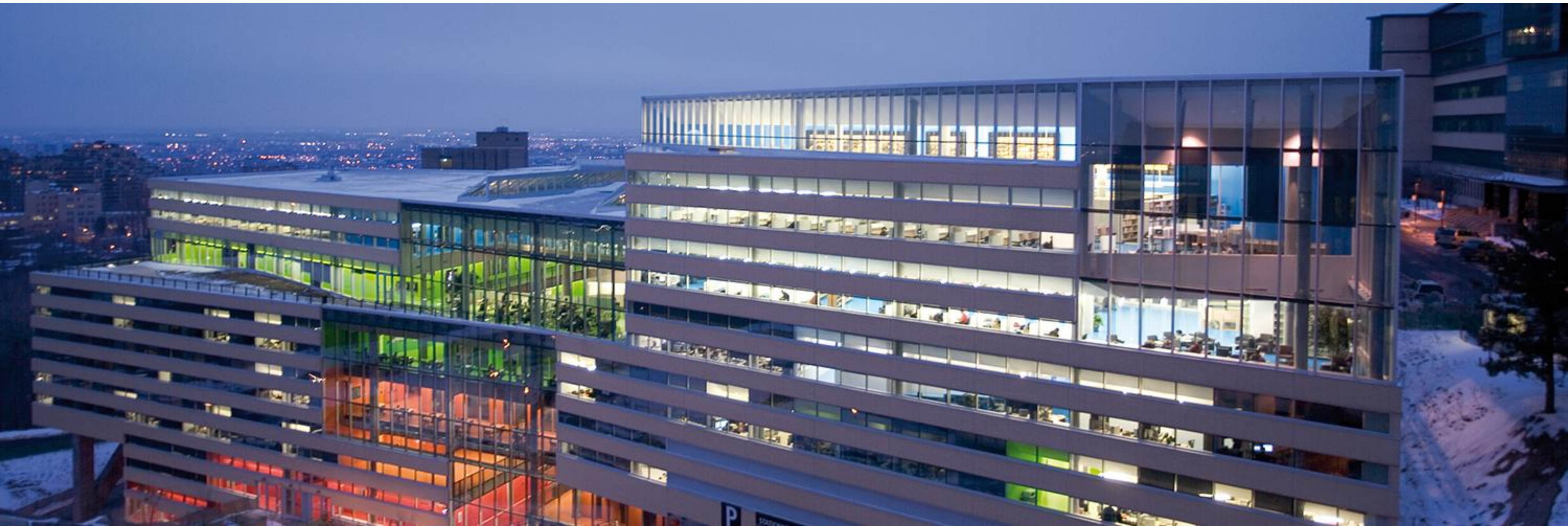
Modern Techniques For Query Evaluation on Highly Connected Datasets

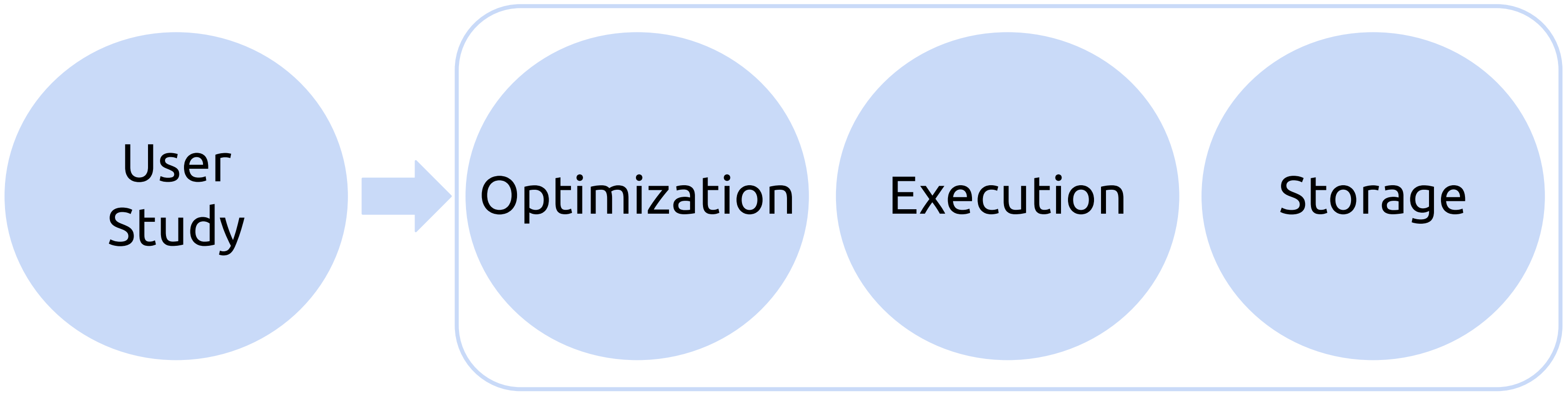
Amine Mhedhbi

Feb. 29th, 2024



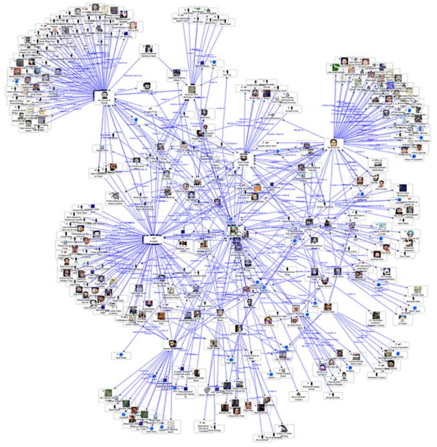
Research Overview

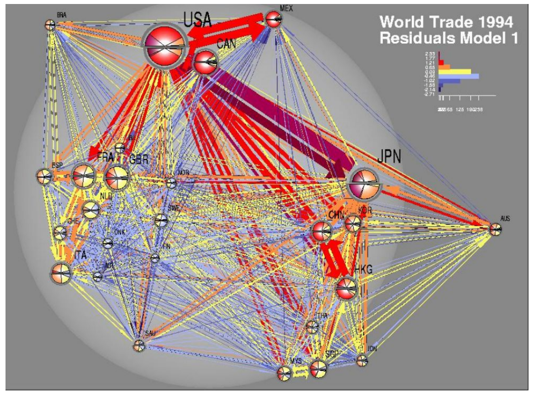
I aim to design and implement data systems capable of *efficient graph data management.*



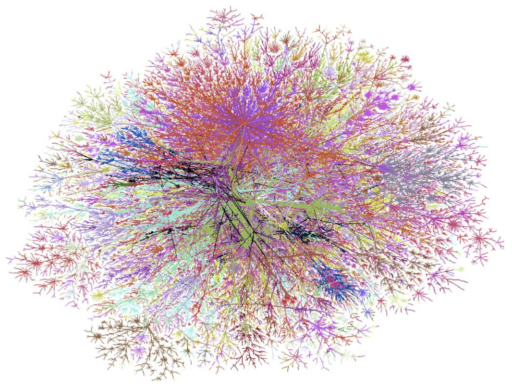
Modeling Application Data As Graph is Ubiquitous

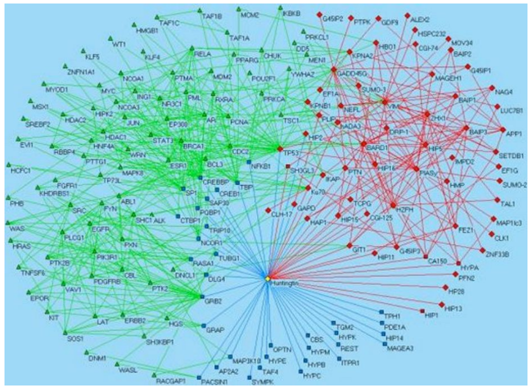
Modeling Application Data As Graph is Ubiquitous

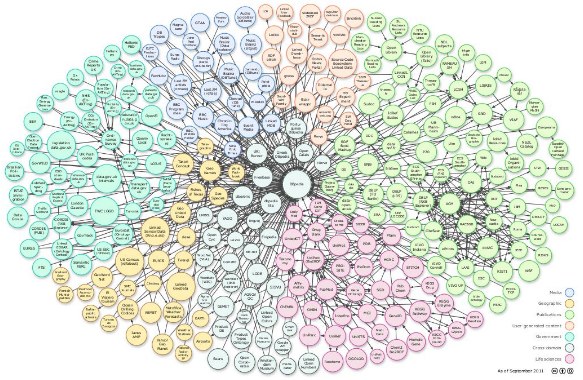

Social Networks

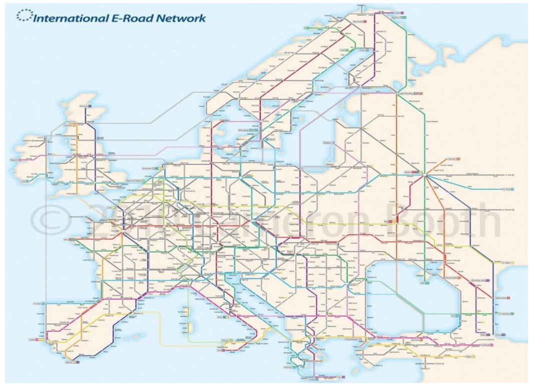


Trade & Financial Connections


Internet / Web

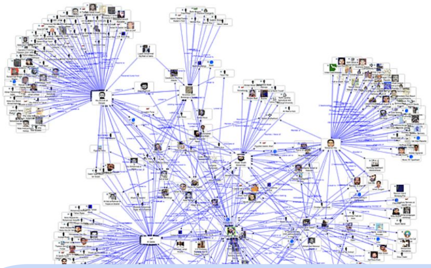

Biological Networks

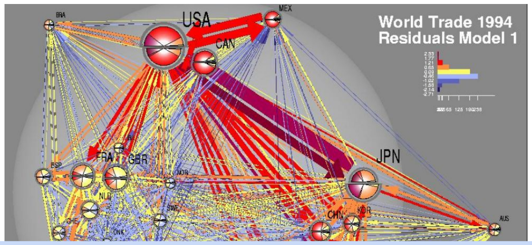

Linked Data *e.g.* Wiki

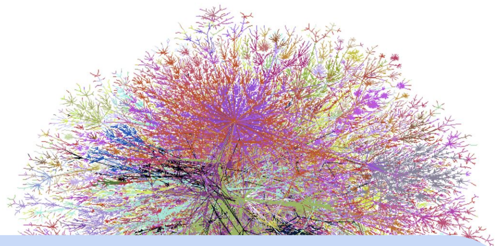


Road Networks

Modeling Application Data As Graph is Ubiquitous



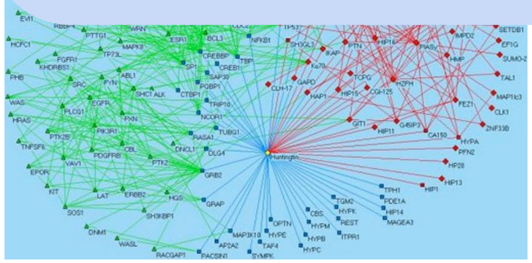


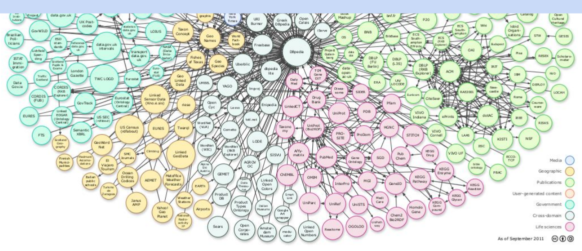


Relationships between entities are central to data analysis

&

Structure of the relationships provides insight



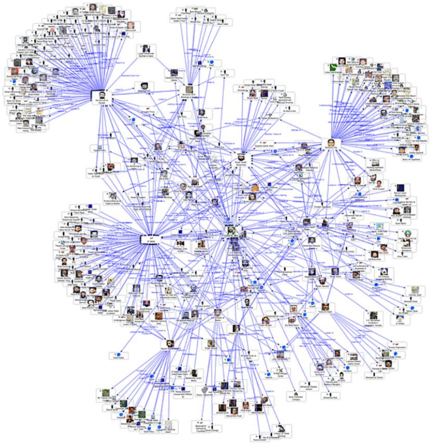


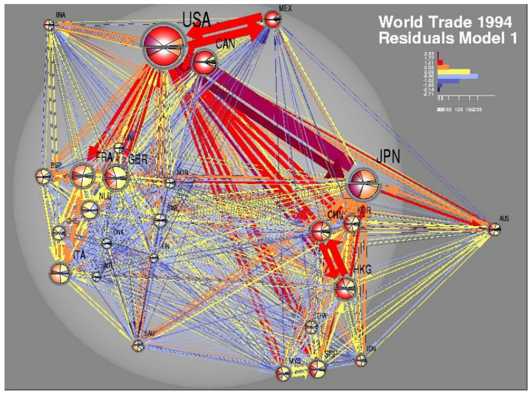
Biological Networks

Linked Data *e.g.* Wiki

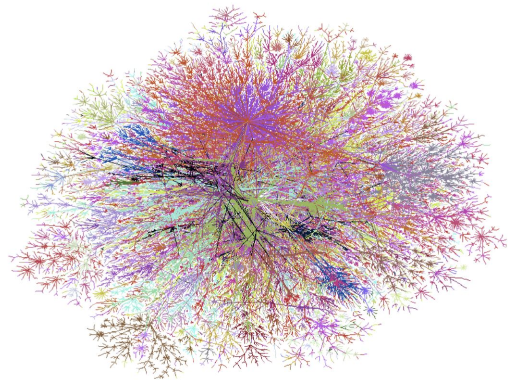
Road Networks

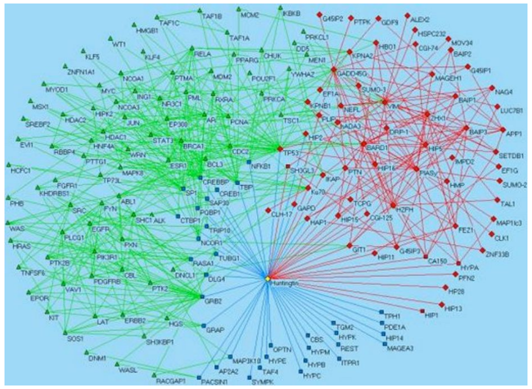
Modeling Application Data As Graph is Ubiquitous

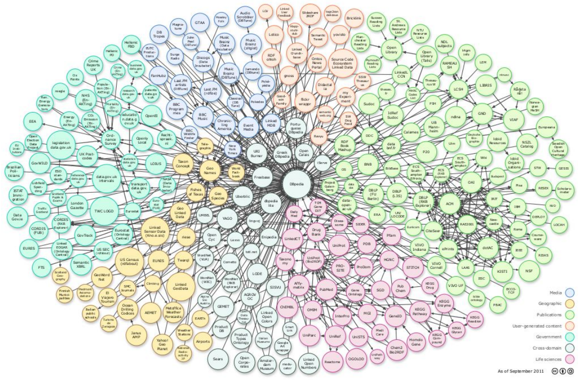

Social Networks

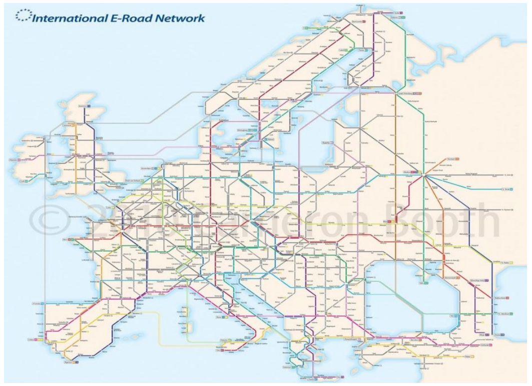


Trade & Financial Connections


Internet / Web


Biological Networks


Linked Data *e.g.* Wiki



Road Networks

Examples of Applications

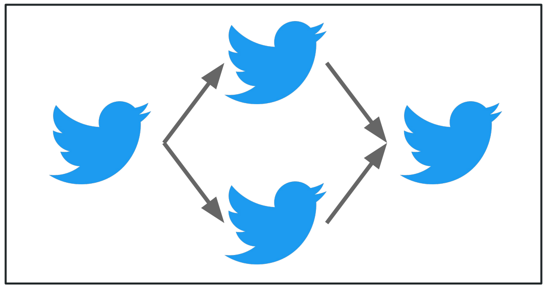
Social Networks

Payment Services

Twitter Recommendation Example

Social Networks

Payment Services

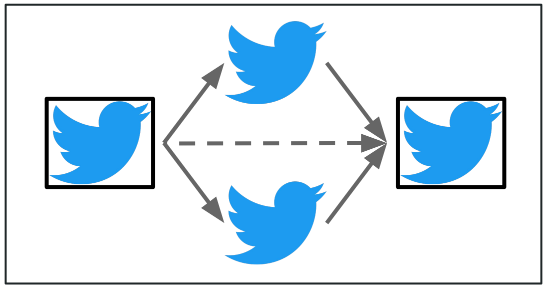


WTF: The Who to Follow Service at Twitter. Gupta et al. WWW 2013.

Twitter Recommendation Example

Social Networks

Payment Services



WTF: The Who to Follow Service at Twitter. Gupta et al. WWW 2013.

Alipay Fraud Detection Example

Social Networks

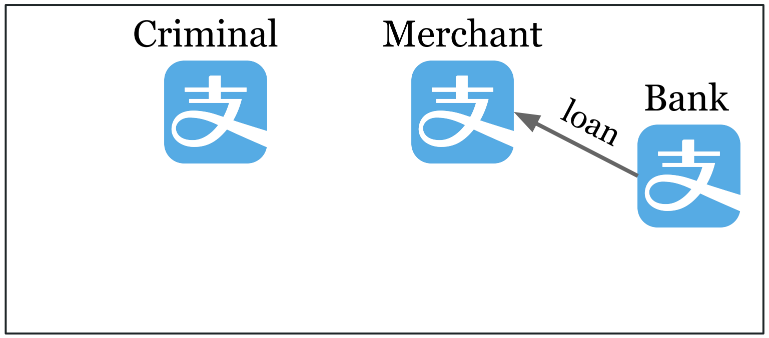
Payment Services

Alipay

Alipay Fraud Detection Example

Social Networks

Payment Services

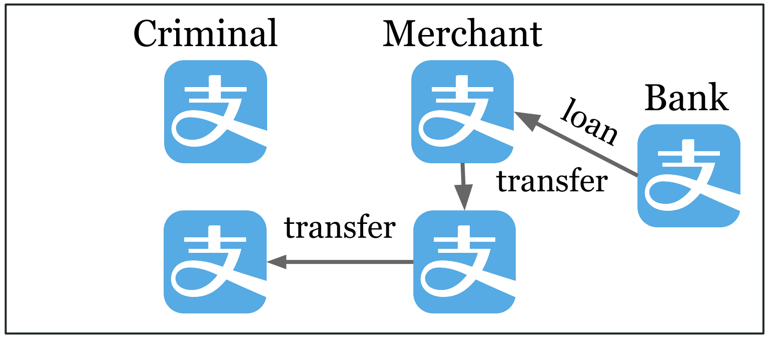


Real-time Constrained Cycle Detection in Large Dynamic Graphs. Qiu et al. VLDB 2018.

Alipay Fraud Detection Example

Social Networks

Payment Services

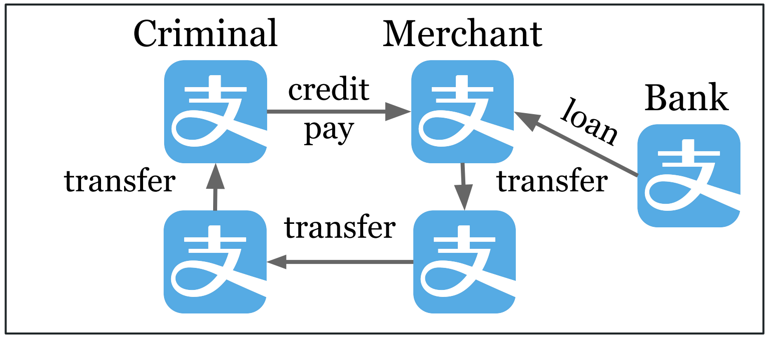


Real-time Constrained Cycle Detection in Large Dynamic Graphs. Qiu et al. VLDB 2018.

Alipay Fraud Detection Example

Social Networks

Payment Services



Real-time Constrained Cycle Detection in Large Dynamic Graphs. Qiu et al. VLDB 2018.

Examples of Applications

Social Networks

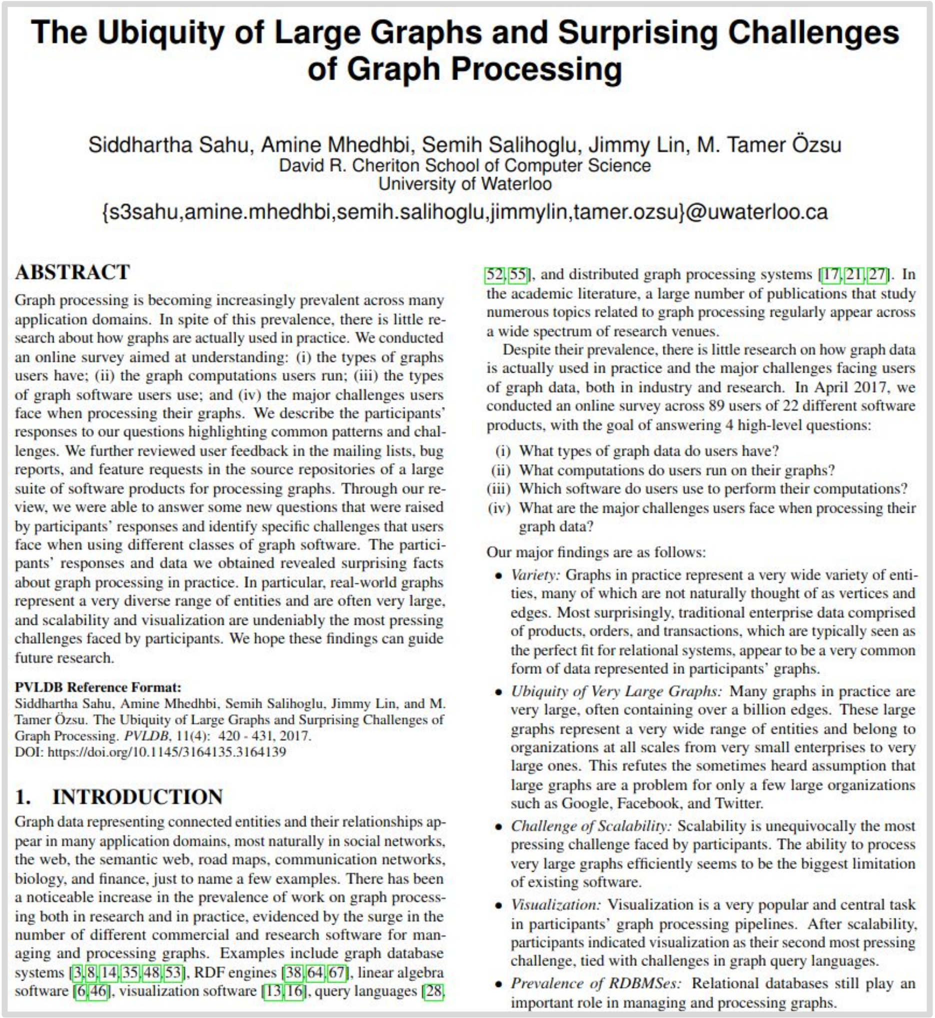
twitter

Payment Services

Alipay

Graph Usage Study

Graph Usage Study



The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing

Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, M. Tamer Özsu
David R. Cheriton School of Computer Science
University of Waterloo
{s3sahu,amine.mhedhbi,semih.salihoglu,jimmylin,tamer.ozsu}@uwaterloo.ca

ABSTRACT

Graph processing is becoming increasingly prevalent across many application domains. In spite of this prevalence, there is little research about how graphs are actually used in practice. We conducted an online survey aimed at understanding: (i) the types of graphs users have; (ii) the graph computations users run; (iii) the types of graph software users use; and (iv) the major challenges users face when processing their graphs. We describe the participants' responses to our questions highlighting common patterns and challenges. We further reviewed user feedback in the mailing lists, bug reports, and feature requests in the source repositories of a large suite of software products for processing graphs. Through our review, we were able to answer some new questions that were raised by participants' responses and identify specific challenges that users face when using different classes of graph software. The participants' responses and data we obtained revealed surprising facts about graph processing in practice. In particular, real-world graphs represent a very diverse range of entities and are often very large, and scalability and visualization are undeniably the most pressing challenges faced by participants. We hope these findings can guide future research.

PVLDB Reference Format:
Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *PVLDB*, 11(4): 420 - 431, 2017.
DOI: https://doi.org/10.1145/3164135.3164139

1. INTRODUCTION

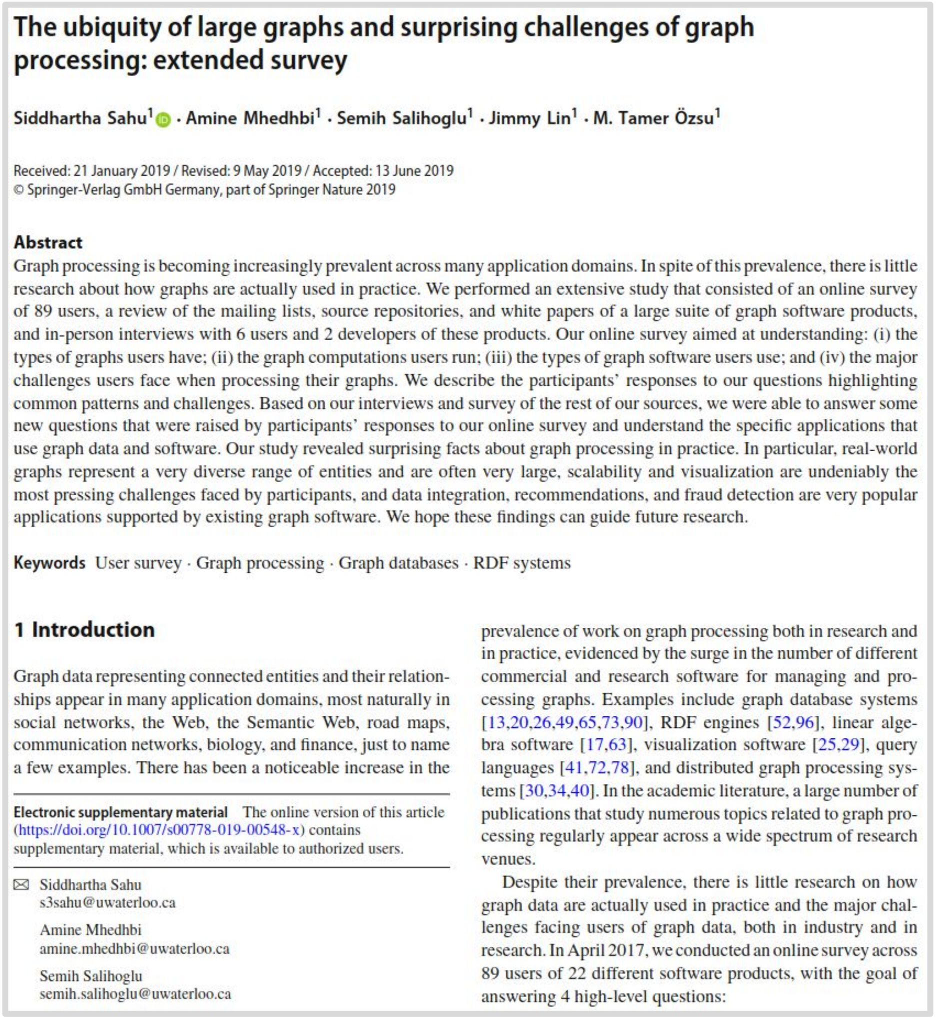
Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the web, the semantic web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [3,8,14,35,48,53], RDF engines [38,64,67], linear algebra software [6,46], visualization software [13,16], query languages [28,52,55], and distributed graph processing systems [17,21,27]. In the academic literature, a large number of publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues.

Despite their prevalence, there is little research on how graph data is actually used in practice and the major challenges facing users of graph data, both in industry and research. In April 2017, we conducted an online survey across 89 users of 22 different software products, with the goal of answering 4 high-level questions:

(i) What types of graph data do users have?
(ii) What computations do users run on their graphs?
(iii) Which software do users use to perform their computations?
(iv) What are the major challenges users face when processing their graph data?

Our major findings are as follows:

- *Variety:* Graphs in practice represent a very wide variety of entities, many of which are not naturally thought of as vertices and edges. Most surprisingly, traditional enterprise data comprised of products, orders, and transactions, which are typically seen as the perfect fit for relational systems, appear to be a very common form of data represented in participants' graphs.
- *Ubiquity of Very Large Graphs:* Many graphs in practice are very large, often containing over a billion edges. These large graphs represent a very wide range of entities and belong to organizations at all scales from very small enterprises to very large ones. This refutes the sometimes heard assumption that large graphs are a problem for only a few large organizations such as Google, Facebook, and Twitter.
- *Challenge of Scalability:* Scalability is unequivocally the most pressing challenge faced by participants. The ability to process very large graphs efficiently seems to be the biggest limitation of existing software.
- *Visualization:* Visualization is a very popular and central task in participants' graph processing pipelines. After scalability, participants indicated visualization as their second most pressing challenge, tied with challenges in graph query languages.
- *Prevalence of RDBMSes:* Relational databases still play an important role in managing and processing graphs.



The ubiquity of large graphs and surprising challenges of graph processing: extended survey

Siddhartha Sahu[1] · Amine Mhedhbi[1] · Semih Salihoglu[1] · Jimmy Lin[1] · M. Tamer Özsu[1]

Received: 21 January 2019 / Revised: 9 May 2019 / Accepted: 13 June 2019

Abstract
Graph processing is becoming increasingly prevalent across many application domains. In spite of this prevalence, there is little research about how graphs are actually used in practice. We performed an extensive study that consisted of an online survey of 89 users, a review of the mailing lists, source repositories, and white papers of a large suite of graph software products, and in-person interviews with 6 users and 2 developers of these products. Our online survey aimed at understanding: (i) the types of graphs users have; (ii) the graph computations users run; (iii) the types of graph software users use; and (iv) the major challenges users face when processing their graphs. We describe the participants' responses to our questions highlighting common patterns and challenges. Based on our interviews and survey of the rest of our sources, we were able to answer some new questions that were raised by participants' responses to our online survey and understand the specific applications that use graph data and software. Our study revealed surprising facts about graph processing in practice. In particular, real-world graphs represent a very diverse range of entities and are often very large, scalability and visualization are undeniably the most pressing challenges faced by participants, and data integration, recommendations, and fraud detection are very popular applications supported by existing graph software. We hope these findings can guide future research.

Keywords User survey · Graph processing · Graph databases · RDF systems

1 Introduction

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the Web, the Semantic Web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [13,20,26,49,65,73,90], RDF engines [52,96], linear algebra software [17,63], visualization software [25,29], query languages [41,72,78], and distributed graph processing systems [30,34,40]. In the academic literature, a large number of publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues.

Despite their prevalence, there is little research on how graph data are actually used in practice and the major challenges facing users of graph data, both in industry and in research. In April 2017, we conducted an online survey across 89 users of 22 different software products, with the goal of answering 4 high-level questions:

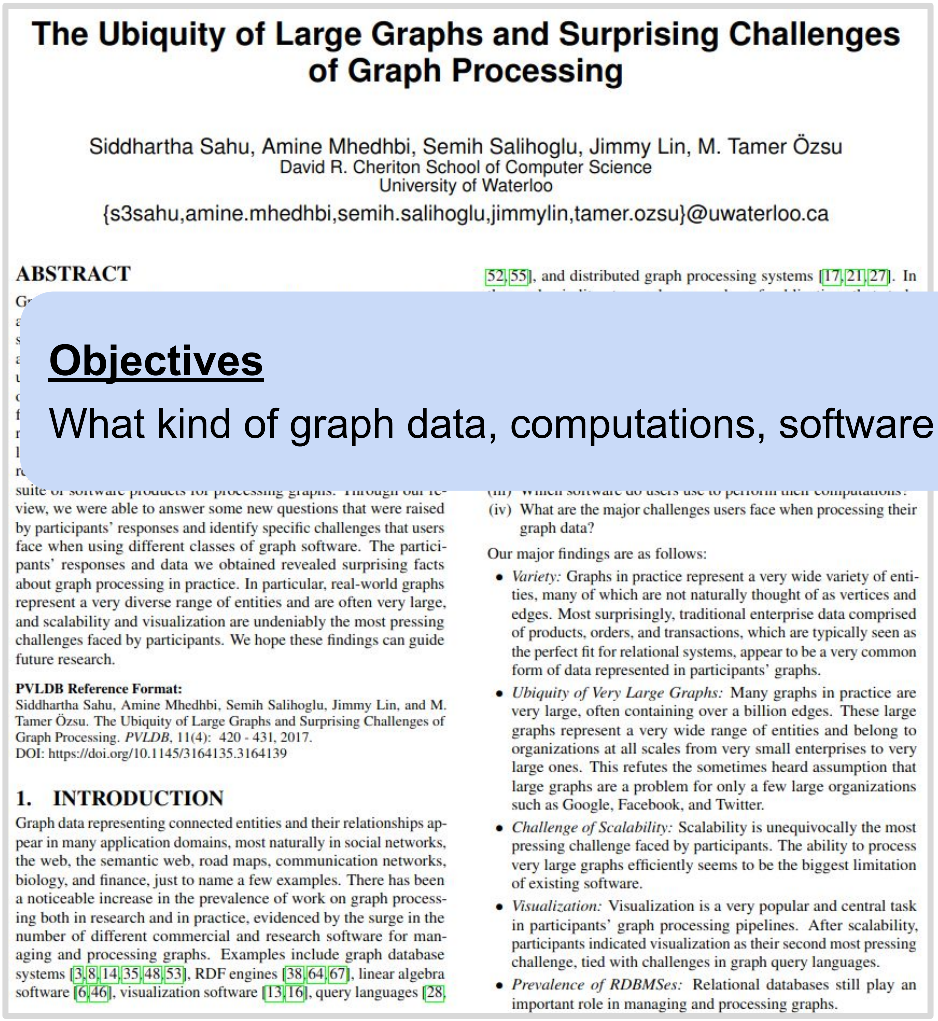
Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-019-00548-x) contains supplementary material, which is available to authorized users.

✉ Siddhartha Sahu
s3sahu@uwaterloo.ca

Amine Mhedhbi
amine.mhedhbi@uwaterloo.ca

Semih Salihoglu
semih.salihoglu@uwaterloo.ca

Graph Usage Study



The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing

Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, M. Tamer Özsu
David R. Cheriton School of Computer Science
University of Waterloo
{s3sahu,amine.mhedhbi,semih.salihoglu,jimmylin,tamer.ozsu}@uwaterloo.ca

ABSTRACT

suite of software products for processing graphs. Through our review, we were able to answer some new questions that were raised by participants' responses and identify specific challenges that users face when using different classes of graph software. The participants' responses and data we obtained revealed surprising facts about graph processing in practice. In particular, real-world graphs represent a very diverse range of entities and are often very large, and scalability and visualization are undeniably the most pressing challenges faced by participants. We hope these findings can guide future research.

PVLDB Reference Format:
Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *PVLDB*, 11(4): 420 - 431, 2017.
DOI: https://doi.org/10.1145/3164135.3164139

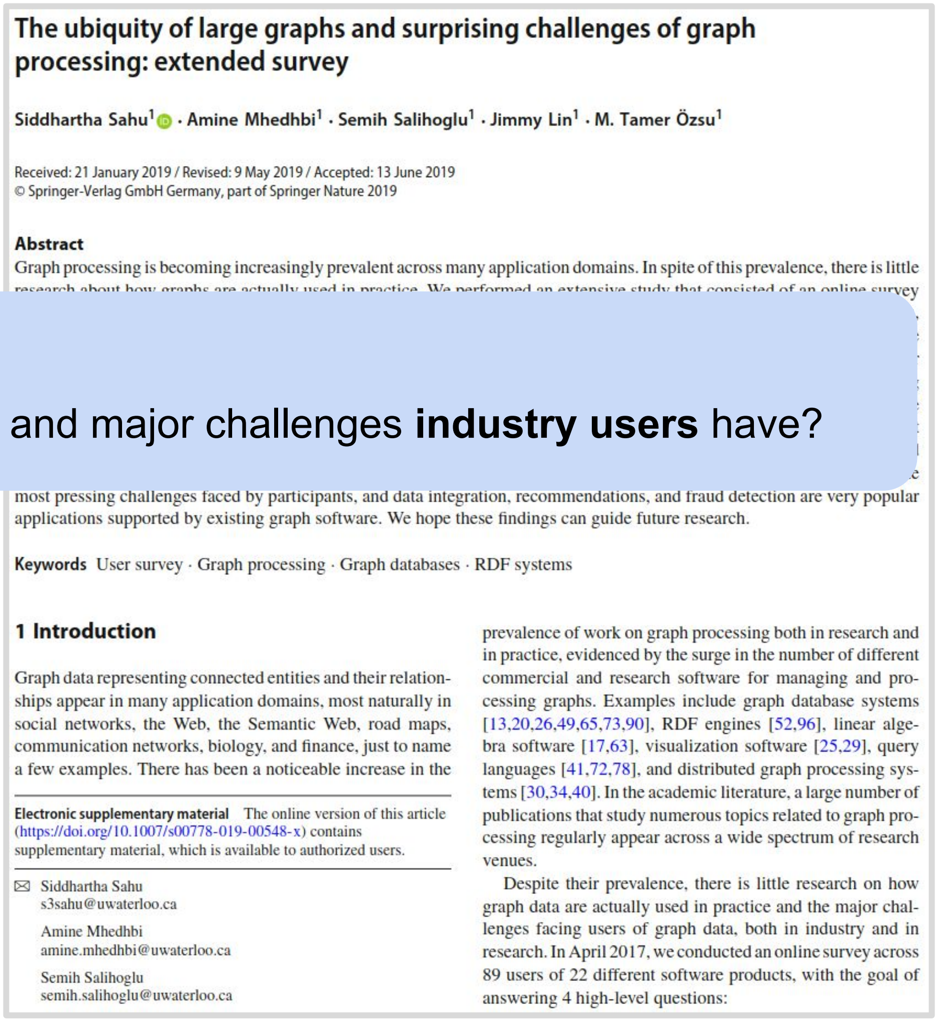
1. INTRODUCTION

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the web, the semantic web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [3,8,14,35,48,53], RDF engines [38,64,67], linear algebra software [6,46], visualization software [13,16], query languages [28, 52,55], and distributed graph processing systems [17,21,27]. In

(iii) Which software do users use to perform their computations?
(iv) What are the major challenges users face when processing their graph data?

Our major findings are as follows:

- *Variety:* Graphs in practice represent a very wide variety of entities, many of which are not naturally thought of as vertices and edges. Most surprisingly, traditional enterprise data comprised of products, orders, and transactions, which are typically seen as the perfect fit for relational systems, appear to be a very common form of data represented in participants' graphs.
- *Ubiquity of Very Large Graphs:* Many graphs in practice are very large, often containing over a billion edges. These large graphs represent a very wide range of entities and belong to organizations at all scales from very small enterprises to very large ones. This refutes the sometimes heard assumption that large graphs are a problem for only a few large organizations such as Google, Facebook, and Twitter.
- *Challenge of Scalability:* Scalability is unequivocally the most pressing challenge faced by participants. The ability to process very large graphs efficiently seems to be the biggest limitation of existing software.
- *Visualization:* Visualization is a very popular and central task in participants' graph processing pipelines. After scalability, participants indicated visualization as their second most pressing challenge, tied with challenges in graph query languages.
- *Prevalence of RDBMSes:* Relational databases still play an important role in managing and processing graphs.



The ubiquity of large graphs and surprising challenges of graph processing: extended survey

Siddhartha Sahu[1] · Amine Mhedhbi[1] · Semih Salihoglu[1] · Jimmy Lin[1] · M. Tamer Özsu[1]

Received: 21 January 2019 / Revised: 9 May 2019 / Accepted: 13 June 2019
© Springer-Verlag GmbH Germany, part of Springer Nature 2019

Abstract

Graph processing is becoming increasingly prevalent across many application domains. In spite of this prevalence, there is little research about how graphs are actually used in practice. We performed an extensive study that consisted of an online survey

most pressing challenges faced by participants, and data integration, recommendations, and fraud detection are very popular applications supported by existing graph software. We hope these findings can guide future research.

Keywords User survey · Graph processing · Graph databases · RDF systems

1 Introduction

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the Web, the Semantic Web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [13,20,26,49,65,73,90], RDF engines [52,96], linear algebra software [17,63], visualization software [25,29], query languages [41,72,78], and distributed graph processing systems [30,34,40]. In the academic literature, a large number of publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues.

Despite their prevalence, there is little research on how graph data are actually used in practice and the major challenges facing users of graph data, both in industry and in research. In April 2017, we conducted an online survey across 89 users of 22 different software products, with the goal of answering 4 high-level questions:

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-019-00548-x) contains supplementary material, which is available to authorized users.

✉ Siddhartha Sahu
s3sahu@uwaterloo.ca

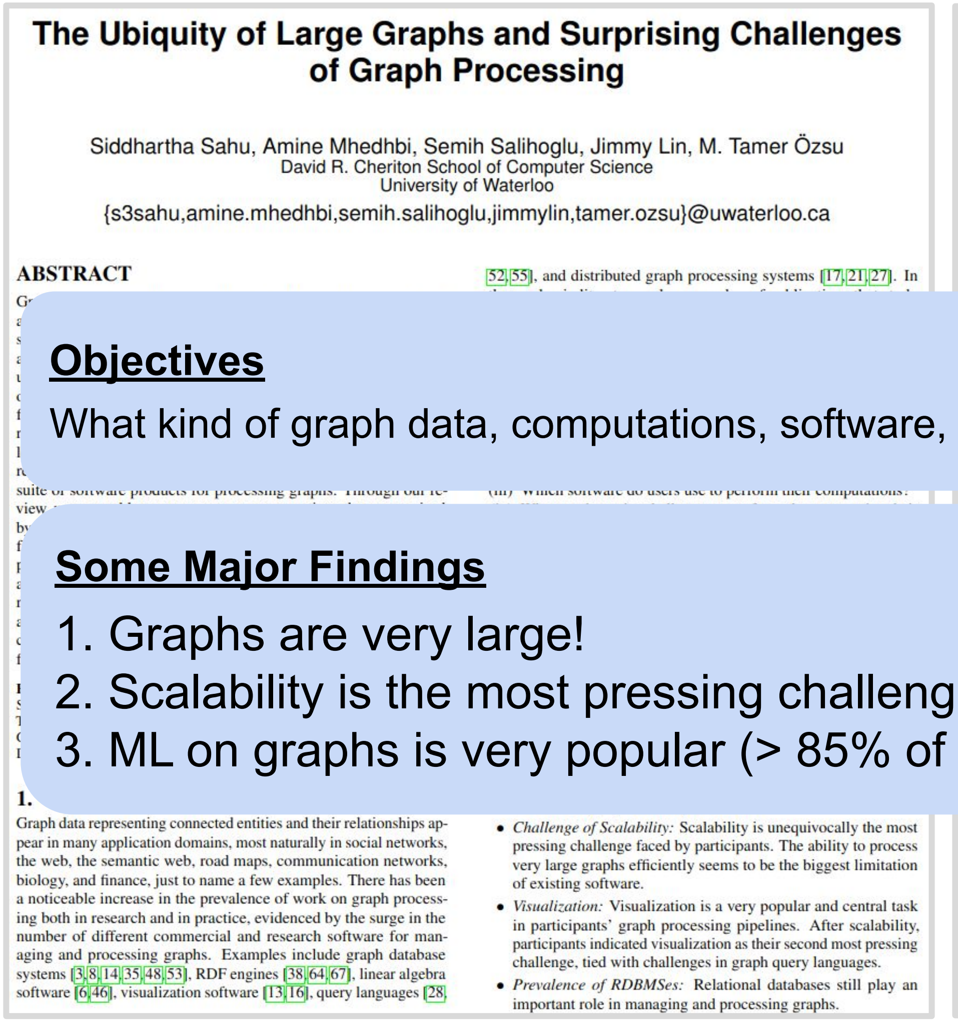
Amine Mhedhbi
amine.mhedhbi@uwaterloo.ca

Semih Salihoglu
semih.salihoglu@uwaterloo.ca

Objectives

What kind of graph data, computations, software, and major challenges **industry users** have?

Graph Usage Study



The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing

Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, M. Tamer Özsu
David R. Cheriton School of Computer Science
University of Waterloo
{s3sahu,amine.mhedhbi,semih.salihoglu,jimmylin,tamer.ozsu}@uwaterloo.ca

ABSTRACT

[52,55], and distributed graph processing systems [17,21,27]. In

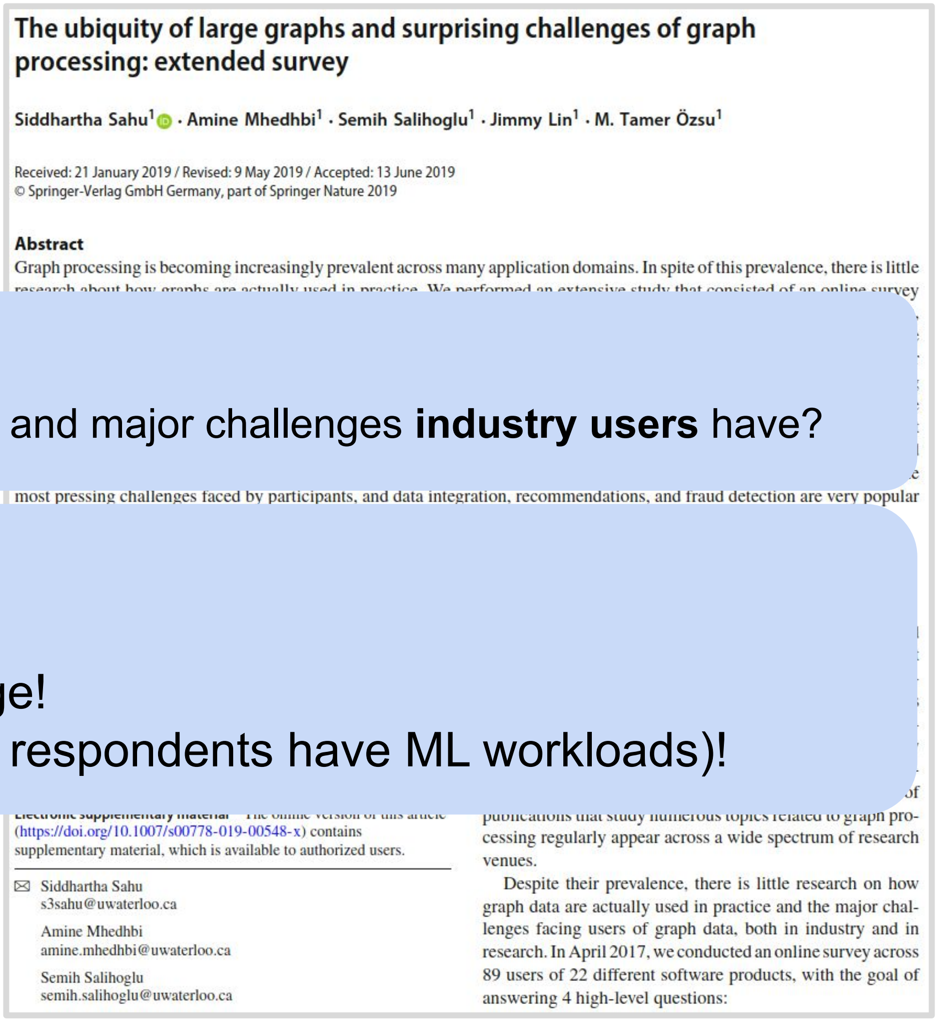
suite of software products for processing graphs. Through our re-
view

(iii) Which software do users use to perform their computations?

1.

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the web, the semantic web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [3,8,14,35,48,53], RDF engines [38,64,67], linear algebra software [6,46], visualization software [13,16], query languages [28,

- *Challenge of Scalability:* Scalability is unequivocally the most pressing challenge faced by participants. The ability to process very large graphs efficiently seems to be the biggest limitation of existing software.
- *Visualization:* Visualization is a very popular and central task in participants' graph processing pipelines. After scalability, participants indicated visualization as their second most pressing challenge, tied with challenges in graph query languages.
- *Prevalence of RDBMSes:* Relational databases still play an important role in managing and processing graphs.



The ubiquity of large graphs and surprising challenges of graph processing: extended survey

Siddhartha Sahu[1] · Amine Mhedhbi[1] · Semih Salihoglu[1] · Jimmy Lin[1] · M. Tamer Özsu[1]

Received: 21 January 2019 / Revised: 9 May 2019 / Accepted: 13 June 2019
© Springer-Verlag GmbH Germany, part of Springer Nature 2019

Abstract
Graph processing is becoming increasingly prevalent across many application domains. In spite of this prevalence, there is little research about how graphs are actually used in practice. We performed an extensive study that consisted of an online survey

most pressing challenges faced by participants, and data integration, recommendations, and fraud detection are very popular

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-019-00548-x) contains supplementary material, which is available to authorized users.

✉ Siddhartha Sahu
s3sahu@uwaterloo.ca

Amine Mhedhbi
amine.mhedhbi@uwaterloo.ca

Semih Salihoglu
semih.salihoglu@uwaterloo.ca

publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues.

Despite their prevalence, there is little research on how graph data are actually used in practice and the major challenges facing users of graph data, both in industry and in research. In April 2017, we conducted an online survey across 89 users of 22 different software products, with the goal of answering 4 high-level questions:

Objectives

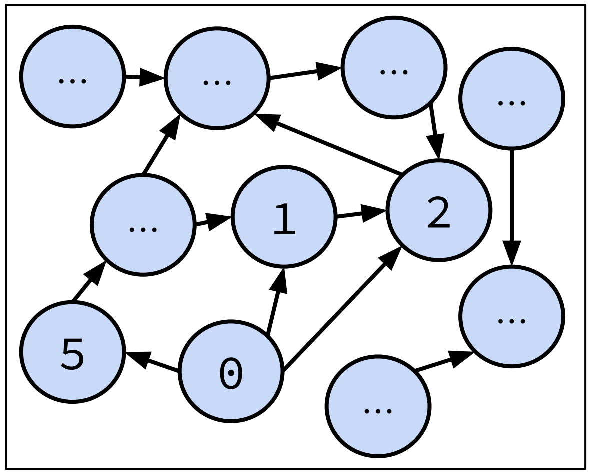
What kind of graph data, computations, software, and major challenges **industry users** have?

Some Major Findings

1. Graphs are very large!
2. Scalability is the most pressing challenge!
3. ML on graphs is very popular (> 85% of respondents have ML workloads)!

Apps Store Facts or Events

Apps Store Facts or Events (Modeling Choice)

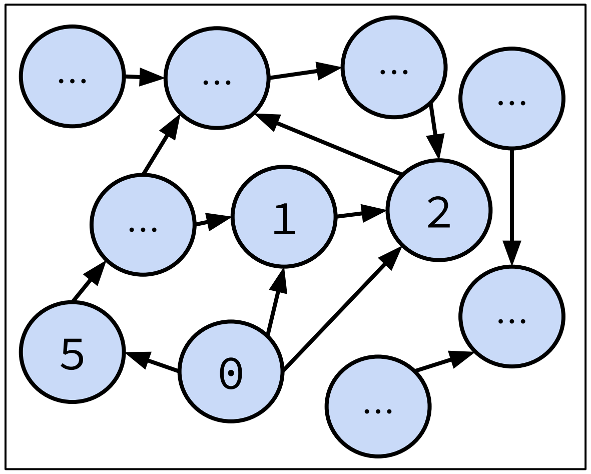


Property Graph Data Model

(FROM, TO)
(0, 1)
(0, 2)
(0, 5)
...

Relational Model

Apps Store Facts or Events (Modeling Choice)



Property Graph Data Model

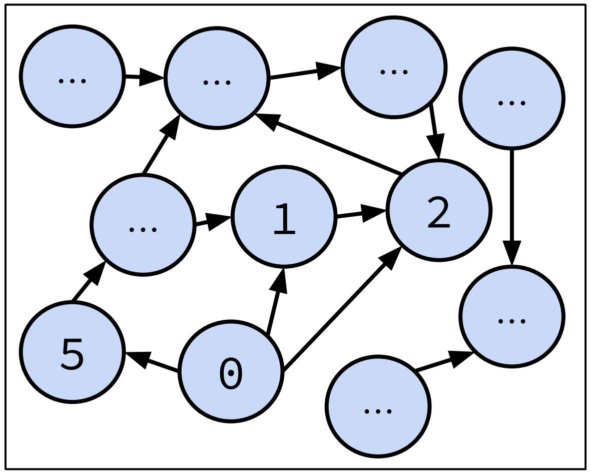
(FROM, TO)
(0, 1)
(0, 2)
(0, 5)
...

Relational Model

Relational algebra primitives!

→ Emphasis on workload i.e., queries and dataset characteristics.

Workload Characteristics

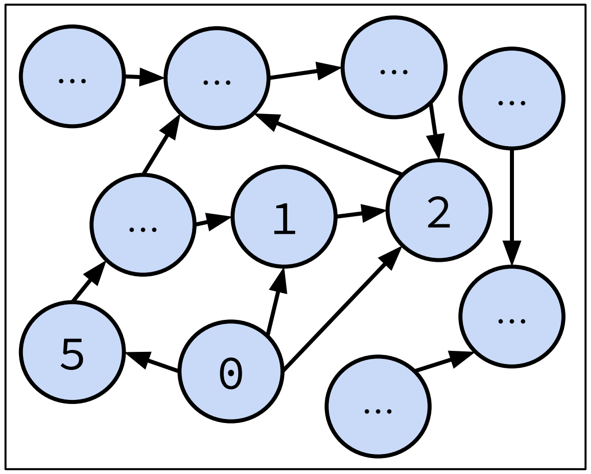


Highly Connected Dataset

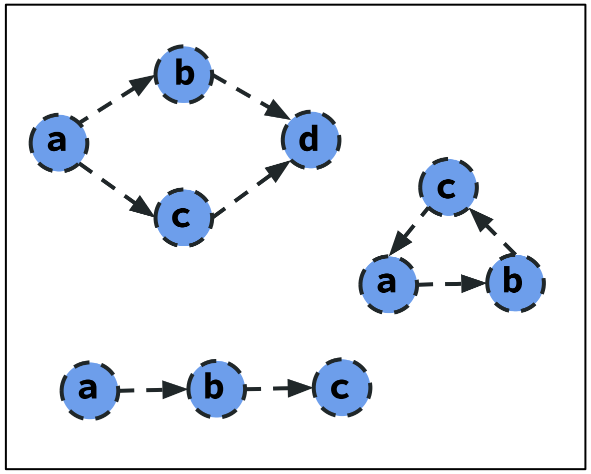
1. Highly Connected data:

   Lots of many-to-many relationships (N-to-M cardinality) !!

Workload Characteristics



Highly Connected Dataset

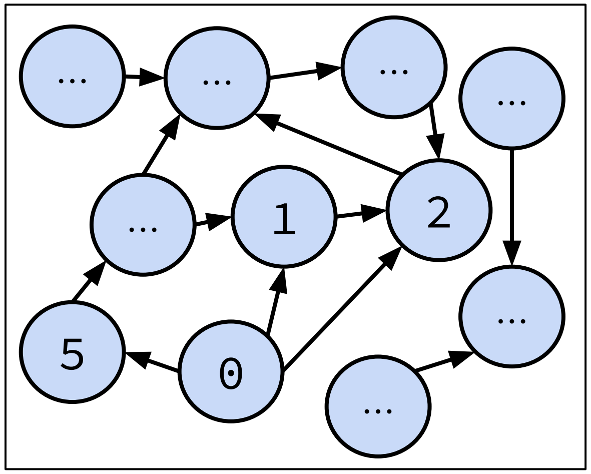


Structure-based Queries

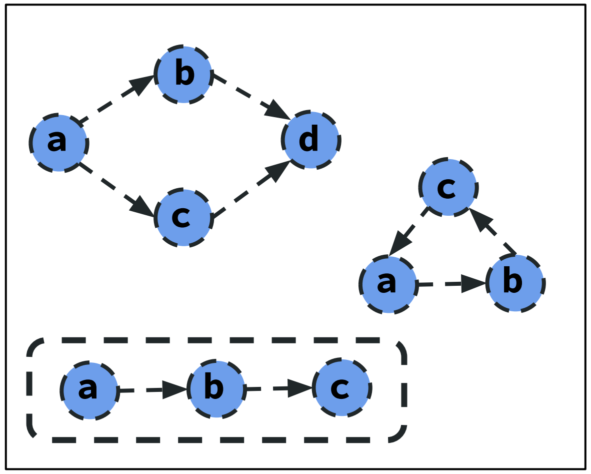
1. Highly Connected data:

   Lots of many-to-many relationships (N-to-M cardinality) !!

Workload Characteristics



Highly Connected Dataset

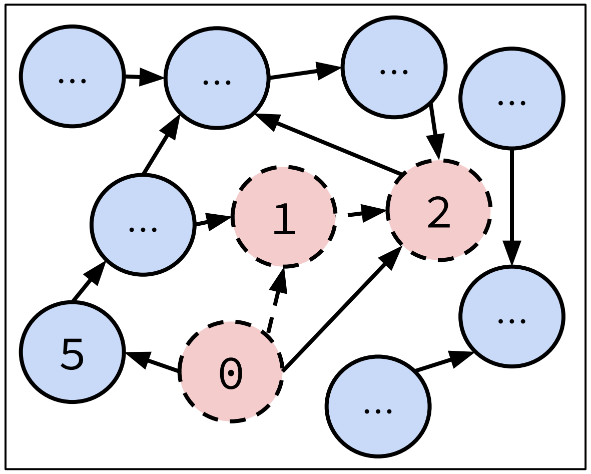


Structure-based Queries

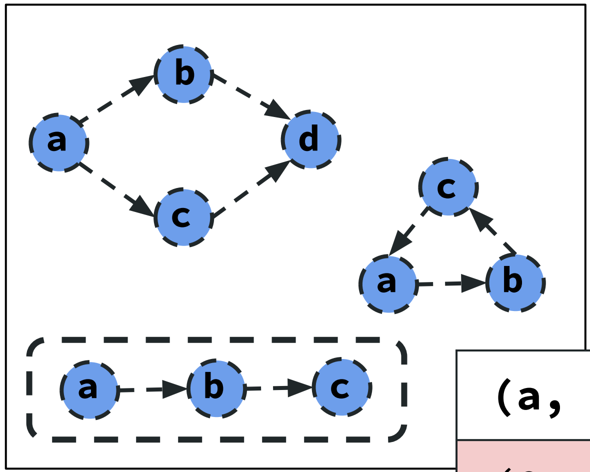
1. <u>Highly Connected data</u>:

   Lots of many-to-many relationships (N-to-M cardinality) !!

Workload Characteristics



Highly Connected Dataset



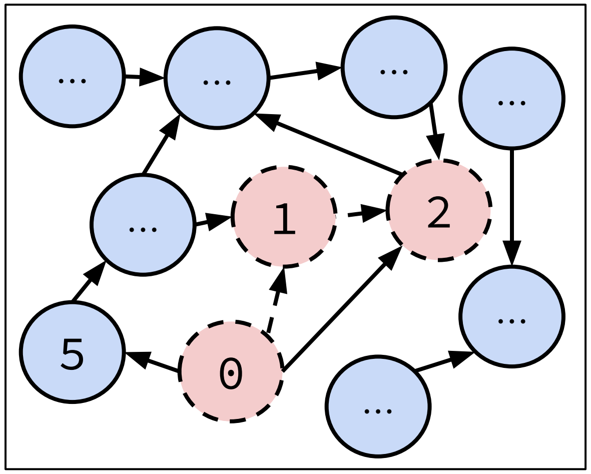
Structure-based Que

(a, b, c)
(0, 1, 2)
…

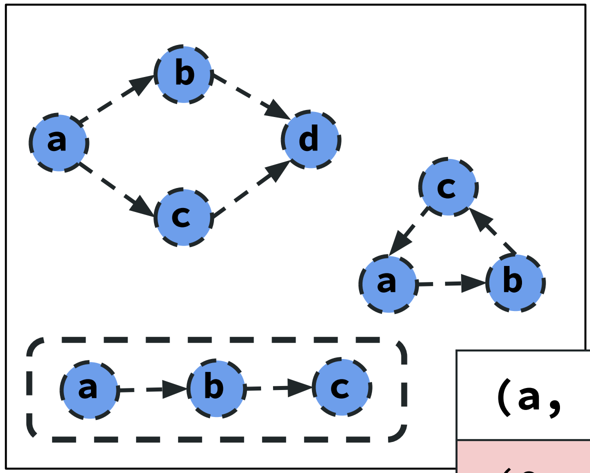
1. Highly Connected data:

   Lots of many-to-many relationships (N-to-M cardinality) !!

Workload Characteristics



Highly Connected Dataset



Structure-based Que

(a, b, c)
(0, 1, 2)
…

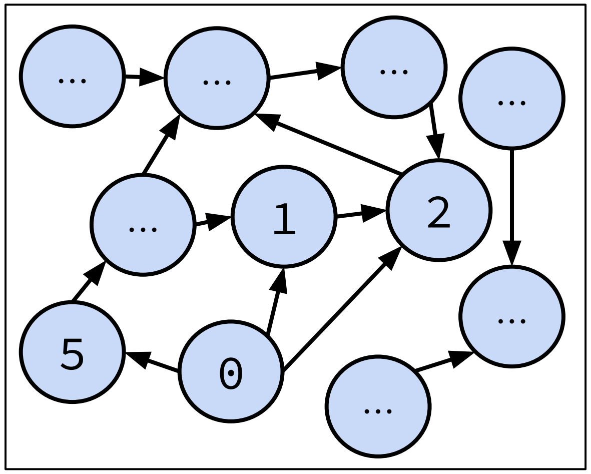
1. <u>Highly Connected data</u>:

   Lots of many-to-many relationships (N-to-M cardinality) !!

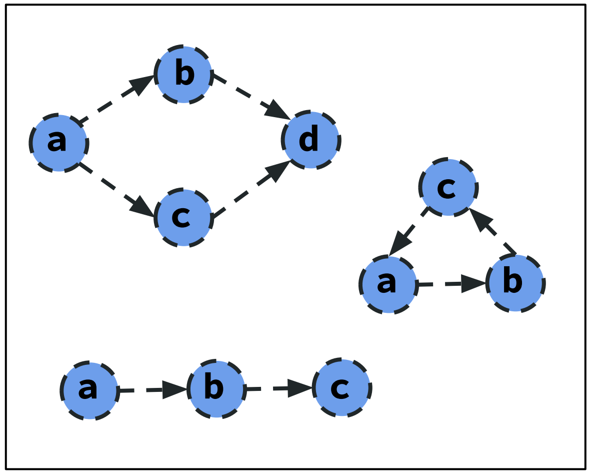
2. <u>Structure-based Queries</u>:

   Queries enumerate graph patterns (*complex many-to-many joins*)

Workload Characteristics



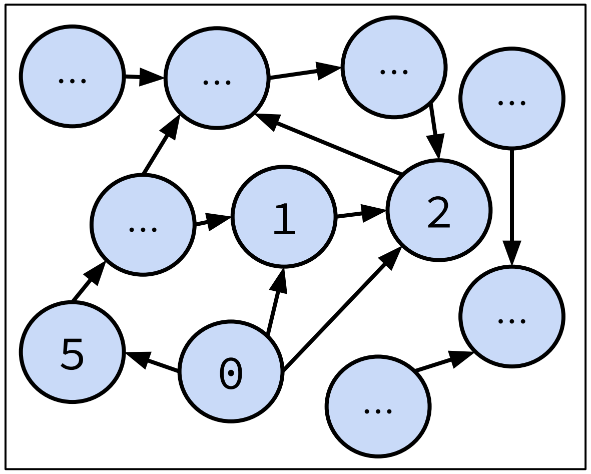
Highly Connected Dataset



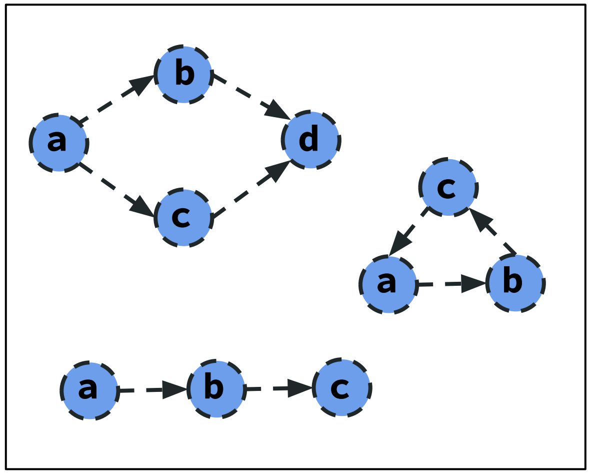
Structure-based Queries

Colloquially called "graph workloads" aka Querying

Workload Characteristics



Highly Connected Dataset



Structure-based Queries

Colloquially called "graph workloads" aka Querying

Interest in ***performance issues***.

Challenging workload due to complex many-to-many joins

→ Existing data systems come short.

Workload Challenges - Many-to-many Joins

Workload Challenges - Many-to-many Joins

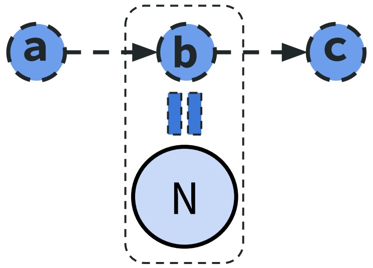
On a financial network, find the accounts whose transactions are facilitated by user with account ID N?

Transactions

src	dst	amount	date

Workload Challenges - Many-to-many Joins

```
On a financial network, find the accounts whose transactions
          are facilitated by user with account ID N?
```



Transactions

src	dst	amount	date

Workload Challenges - Many-to-many Joins

```
On a financial network, find the accounts whose transactions
         are facilitated by user with account ID N?
```

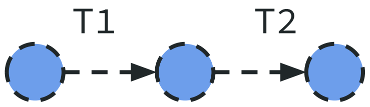
Transactions

src	dst	amount	date

```
 FROM Transactions T1, Transactions T2
WHERE T1.dst = T2.src
```

Workload Challenges - Many-to-many Joins

On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



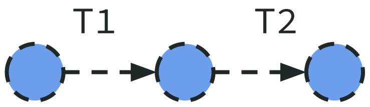
Transactions

src	dst	amount	date

```
 FROM Transactions T1, Transactions T2
WHERE T1.dst = T2.src
```

Workload Challenges - Many-to-many Joins

On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



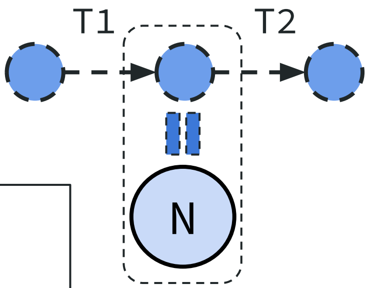
Transactions

src	dst	amount	date

```
 FROM Transactions T1, Transactions T2
WHERE T1.dst = T2.src
  AND T1.dst = N
```

Workload Challenges - Many-to-many Joins

On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



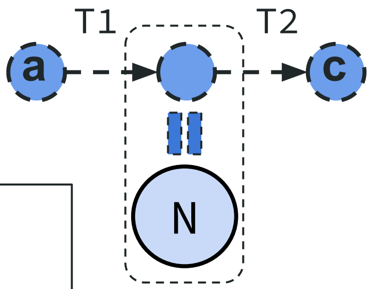
Transactions

src	dst	amount	date

```
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```

Workload Challenges - Many-to-many Joins

On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



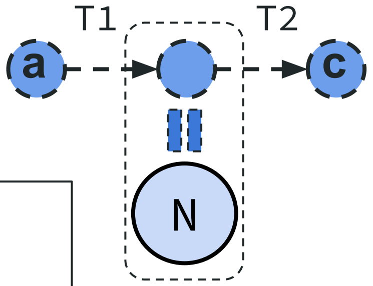
Transactions

src	dst	amount	date

```
SELECT T1.src as a, T2.dst as c
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```

Workload Challenges - Many-to-many Joins

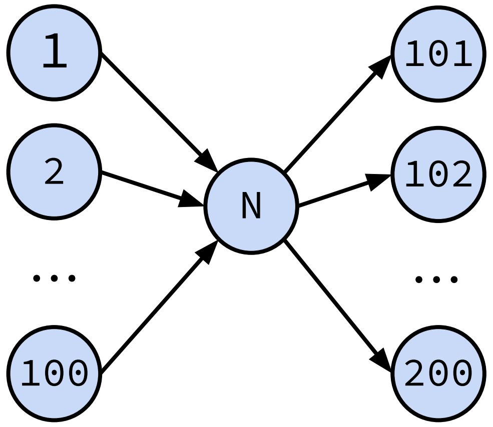
On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



Transactions

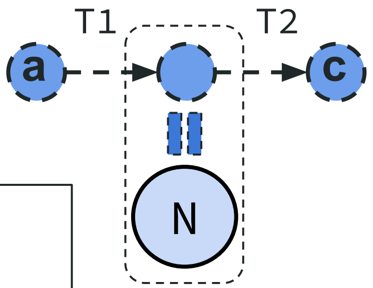
src	dst	amount	date

```
SELECT T1.src as a, T2.dst as c
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```



Workload Challenges - Many-to-many Joins

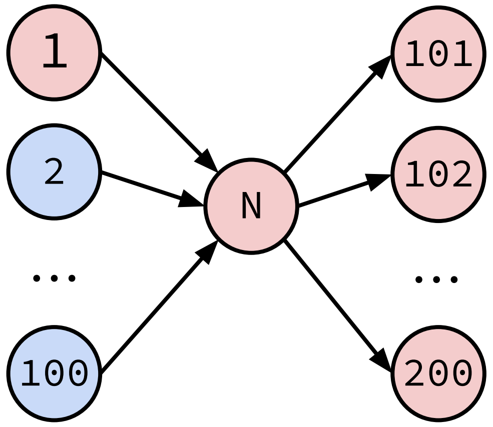
On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



Transactions

src	dst	amount	date

```
SELECT T1.src as a, T2.dst as c
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```

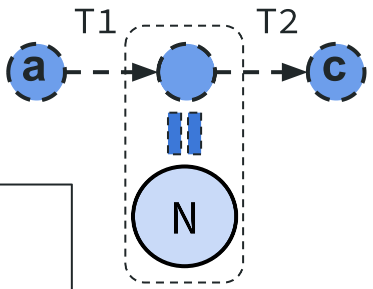


(a, c)
(1, 101)
(1, ...)
(1, 200)

100 tuples

Workload Challenges - Many-to-many Joins

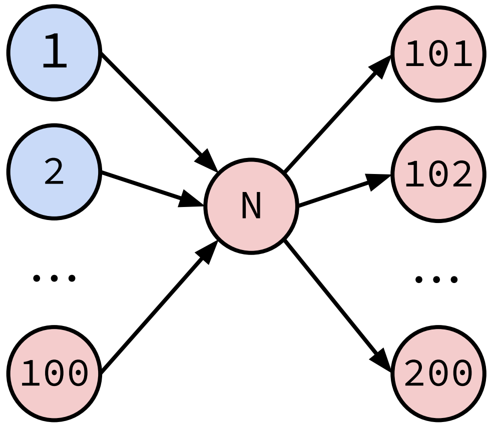
On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



Transactions

src	dst	amount	date

```
SELECT T1.src as a, T2.dst as c
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```

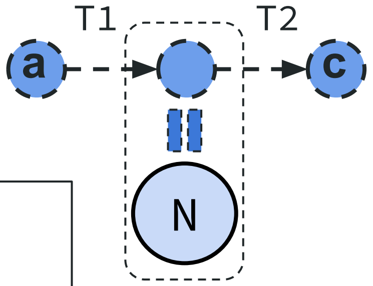


(a, c)
(1, 101)
(1, ...)
(1, 200)
...
(100, 101)
(100, ...)
(100, 200)

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ 100 tuples

Workload Challenges - Many-to-many Joins

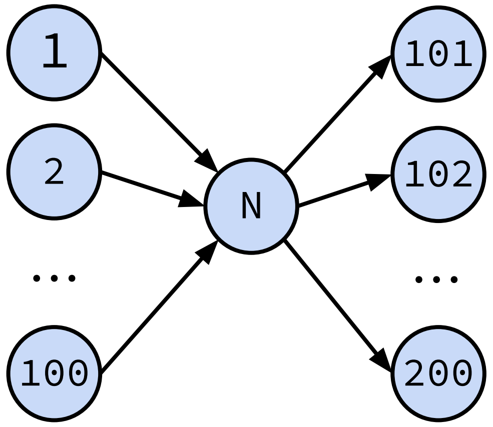
On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



Transactions

src	dst	amount	date

```
SELECT T1.src as a, T2.dst as c
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```

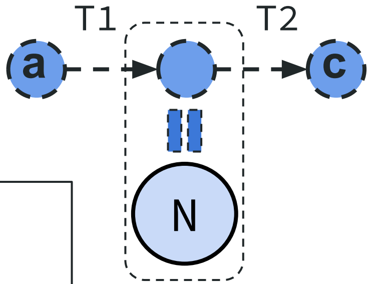


(a, c)
(1, 101)
(1, ...)
(1, 200)
...
(100, 101)
(100, ...)
(100, 200)

200 edges → 10,000 !!!

Workload Challenges - Many-to-many Joins

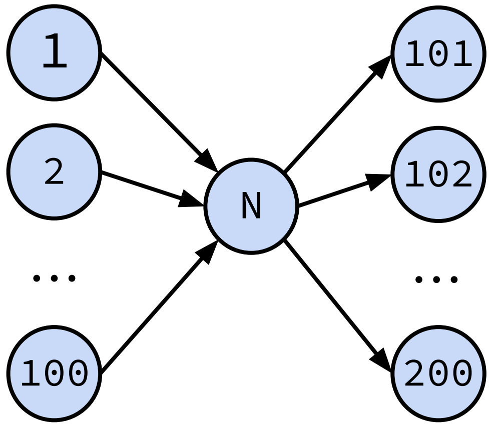
On a financial network, find the accounts whose transactions are facilitated by user with account ID N?



Transactions

src	dst	amount	date

```
SELECT T1.src as a, T2.dst as c
  FROM Transactions T1, Transactions T2
 WHERE T1.dst = T2.src
   AND T1.dst = N
```



(a, c)
(1, 101)
(1, ...)
(1, 200)
...
(100, 101)
(100, ...)
(100, 200)

200 edges → 10,000 !!!

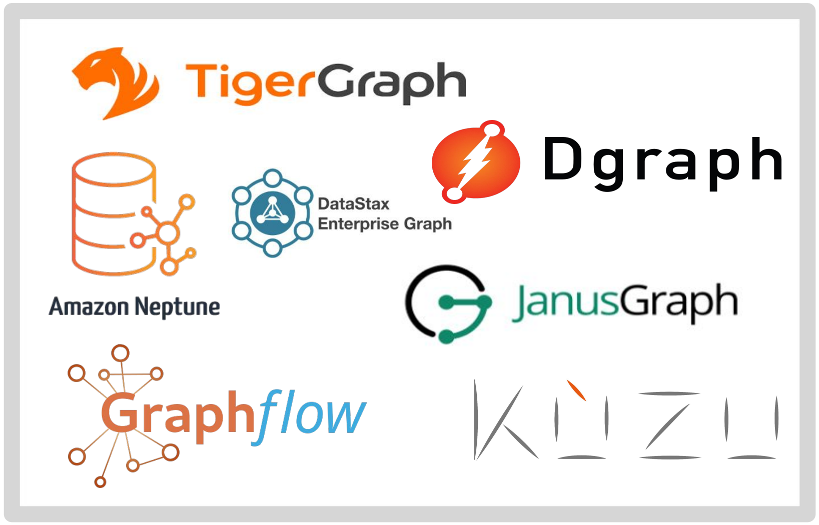
Explosion in intermediate results size!!!

DBMSs Evaluating Graph Workloads

Analytical RDBMSs do not optimize for graph workloads

DBMSs Evaluating Graph Workloads

Analytical RDBMSs do not optimize for graph workloads

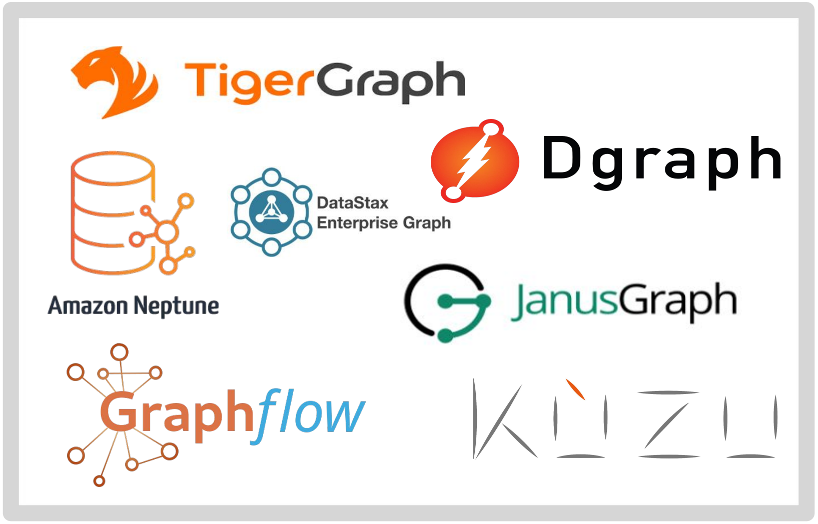


New graph DBMSs (GDBMSs) optimize for graph workloads

DBMSs Evaluating Graph Workloads

teradata.
IBM DB2
Microsoft SQL Server
Amazon RDS
SQL
ORACLE DATABASE
MySQL
SQL
snowflake
PostgreSQL
SAP S/4 HANA

Analytical RDBMSs do not optimize for graph workloads

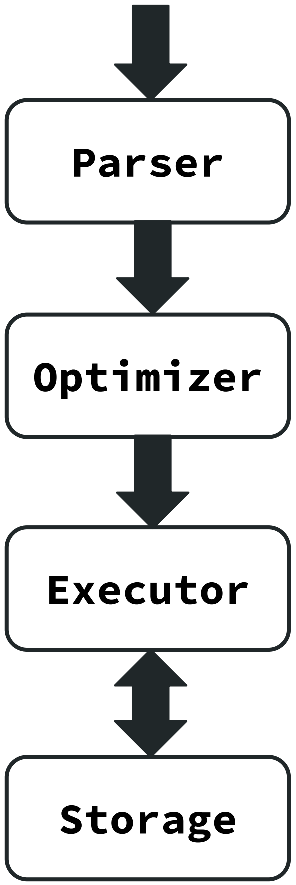


New graph DBMSs (GDBMSs) optimize for graph workloads

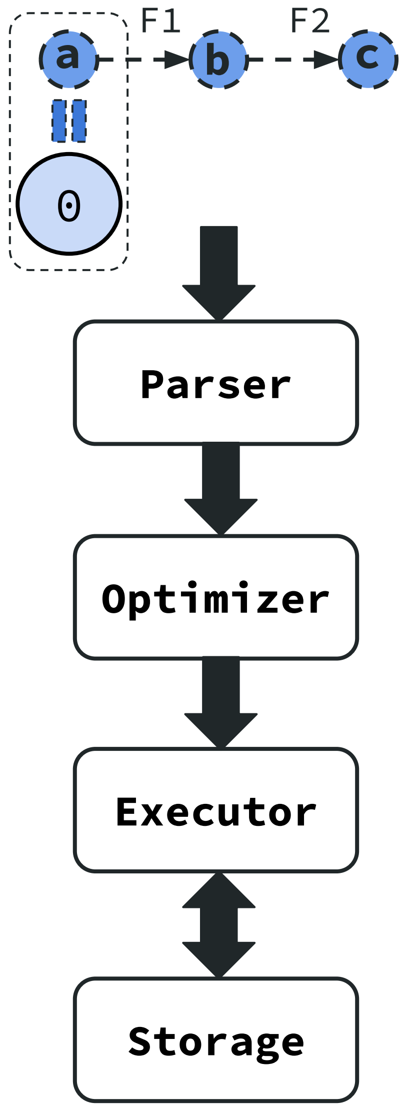
High-level Research Question – *How should Database Management Systems (DBMSs) be architected to optimize for analytical graph workloads?*

Revisiting DBMS Components

Query in Declarative Language e.g., SQL, Cypher, etc.

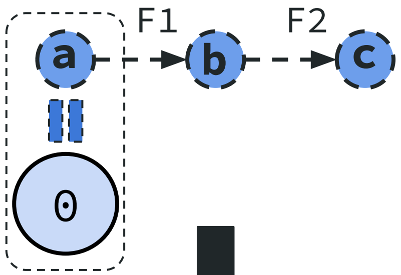


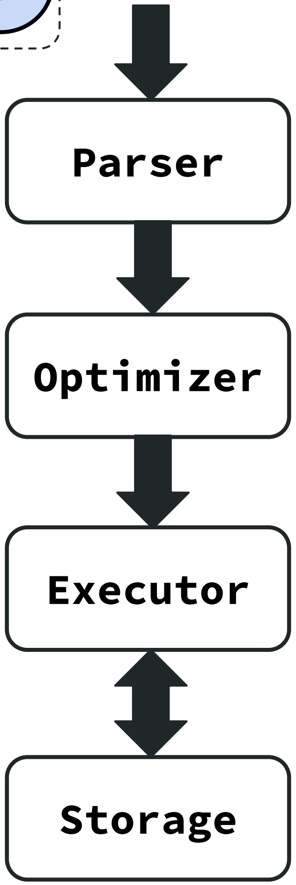
Revisiting DBMS Components (Example)



Revisiting DBMS Components (Example)

Cypher

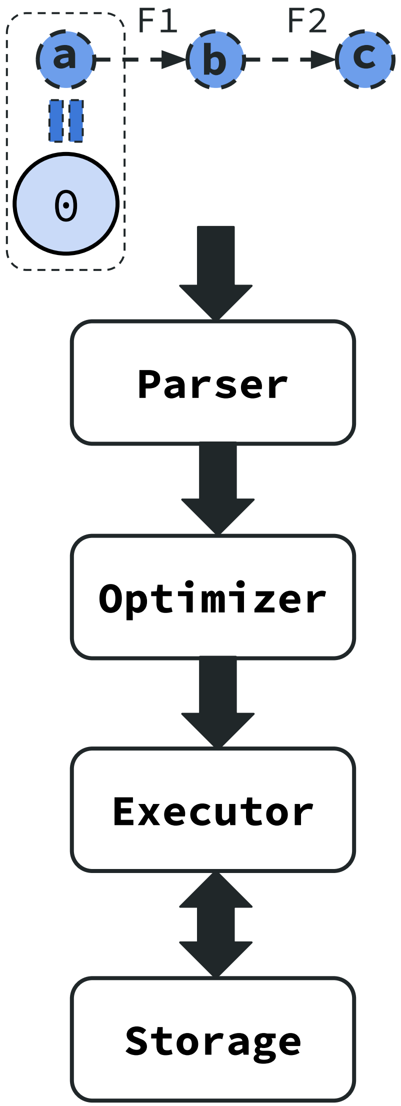




Revisiting DBMS Components (Example)

Cypher

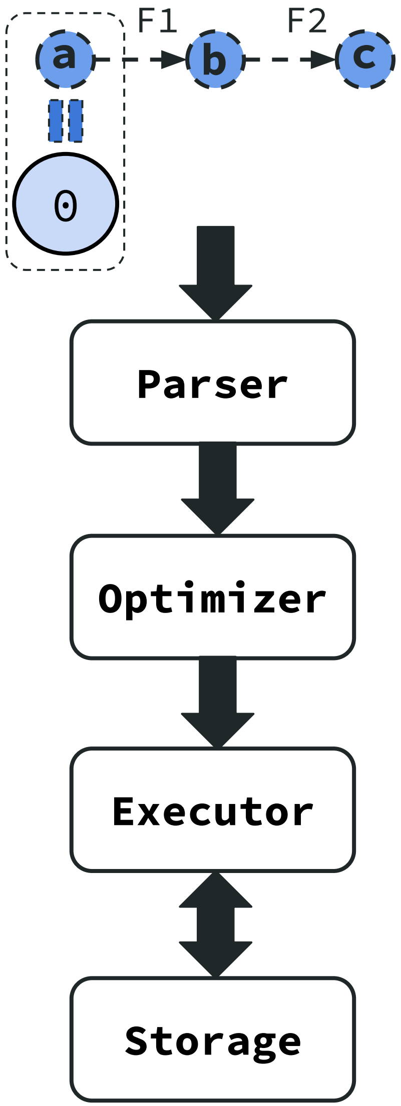
```
MATCH (a)-[Follows]->(b),
      (b)-[Follows]->(c)
```



Revisiting DBMS Components (Example)

Cypher

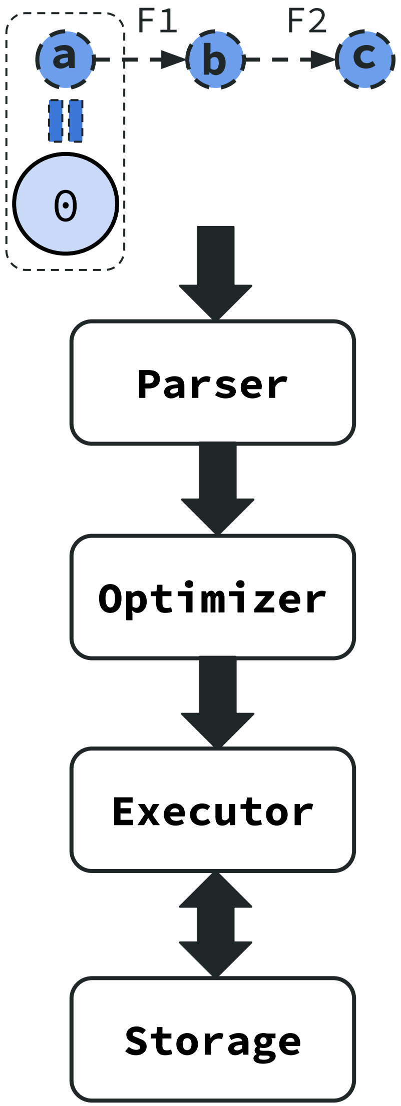
```
MATCH (a)-[Follows]->(b),
      (b)-[Follows]->(c)
WHERE  a.ID = 0
```



Revisiting DBMS Components (Example)

Cypher

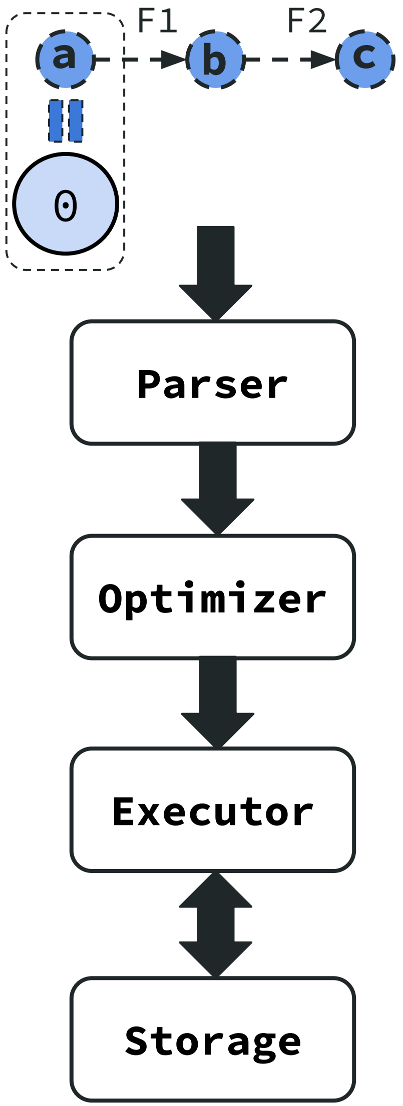
```
MATCH (a)-[Follows]->(b),
      (b)-[Follows]->(c)
WHERE  a.ID = 0
RETURN b.ID, c.ID
```



Revisiting DBMS Components (Example)

Cypher

```
MATCH (a)-[Follows]->(b),
      (b)-[Follows]->(c)
WHERE  a.ID = 0
RETURN b.ID, c.ID
```

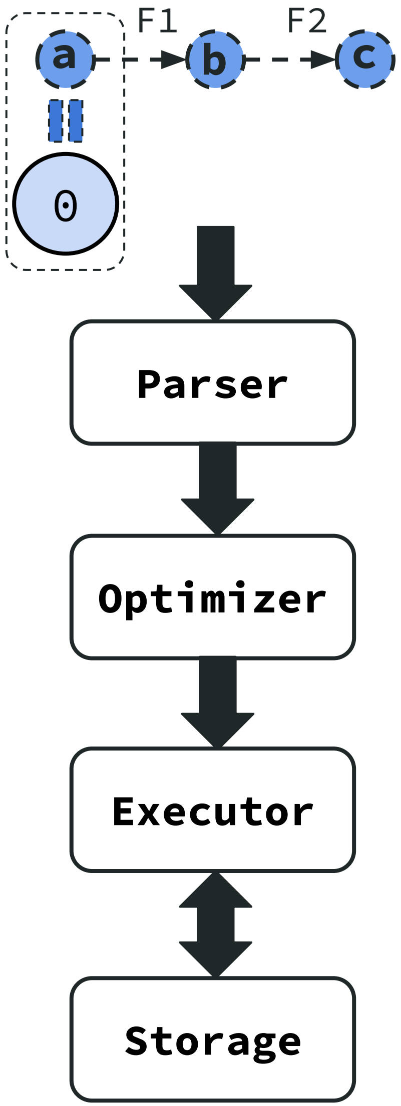


SQL

Revisiting DBMS Components (Example)

Cypher

```
MATCH (a)-[Follows]->(b),
      (b)-[Follows]->(c)
WHERE  a.ID = 0
RETURN b.ID, c.ID
```



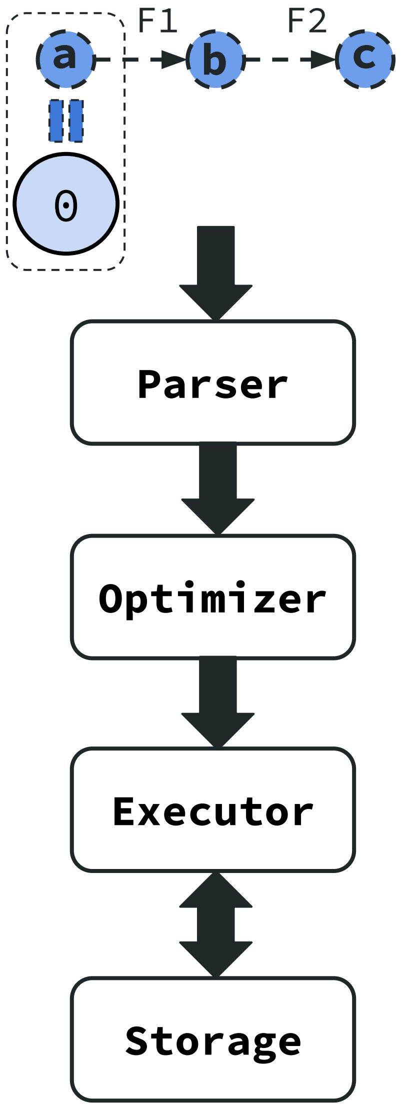
SQL

```
FROM    Follows F1,
        Follows F2
```

Revisiting DBMS Components (Example)

Cypher

```
MATCH  (a)-[Follows]->(b),
       (b)-[Follows]->(c)
WHERE   a.ID = 0
RETURN b.ID, c.ID
```



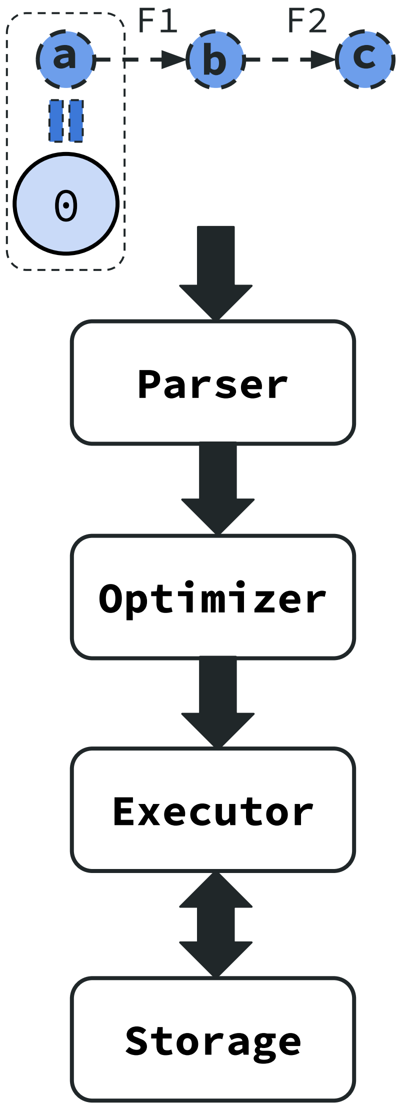
SQL

```



FROM     Follows F1,
         Follows F2
WHERE    F1.FROM = 0
  AND    F1.TO = F2.FROM
```

Revisiting DBMS Components (Example)

Cypher

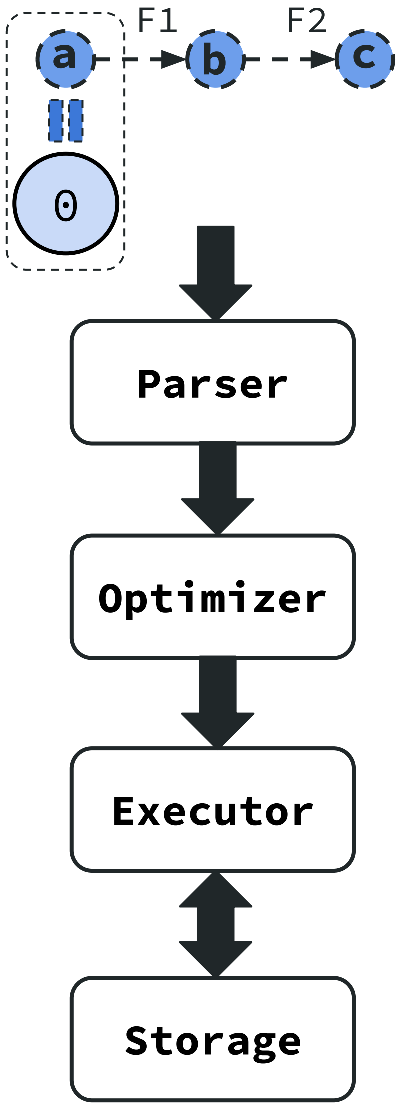
```
MATCH  (a)-[Follows]->(b),
       (b)-[Follows]->(c)
WHERE   a.ID = 0
RETURN b.ID, c.ID
```



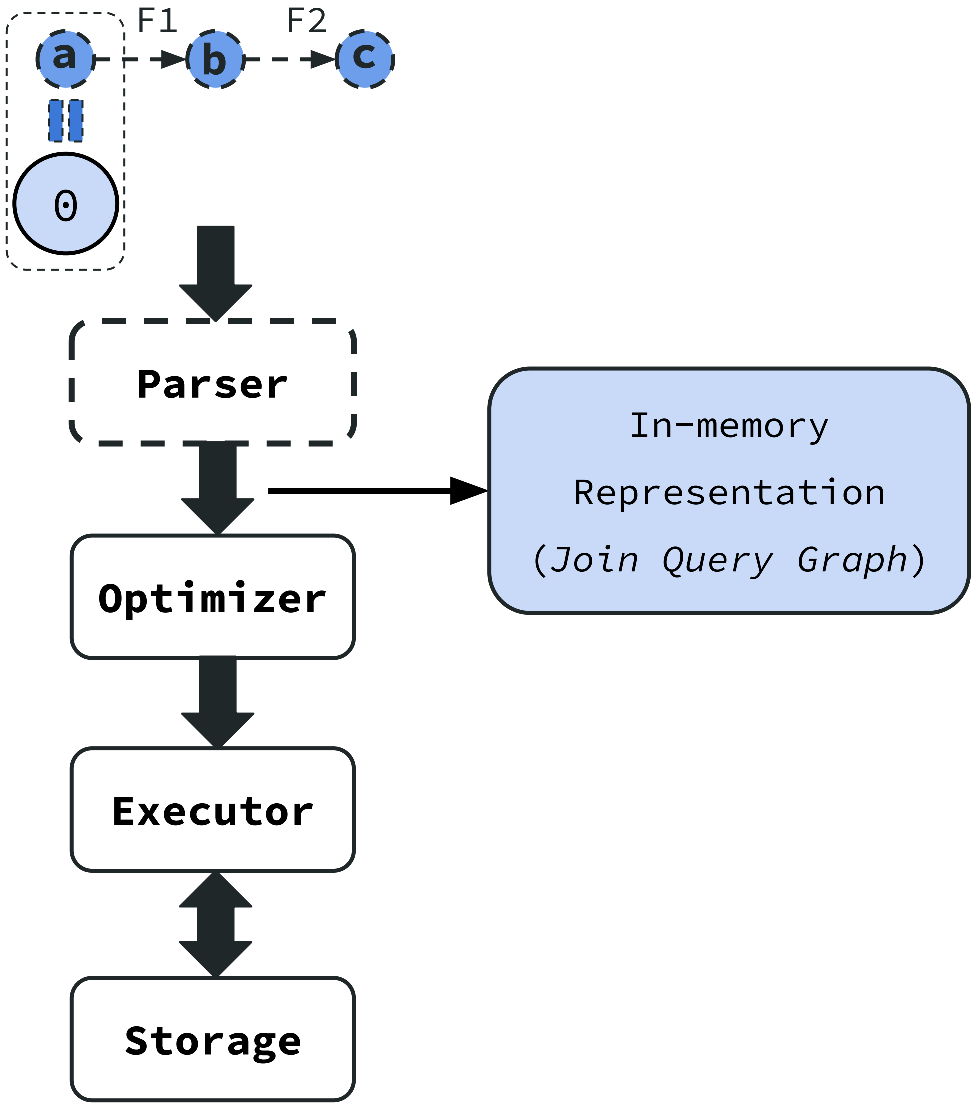
SQL

```
SELECT  F2.FROM as b,
        F2.TO   as c
FROM    Follows F1,
        Follows F2
WHERE   F1.FROM = 0
  AND   F1.TO = F2.FROM
```

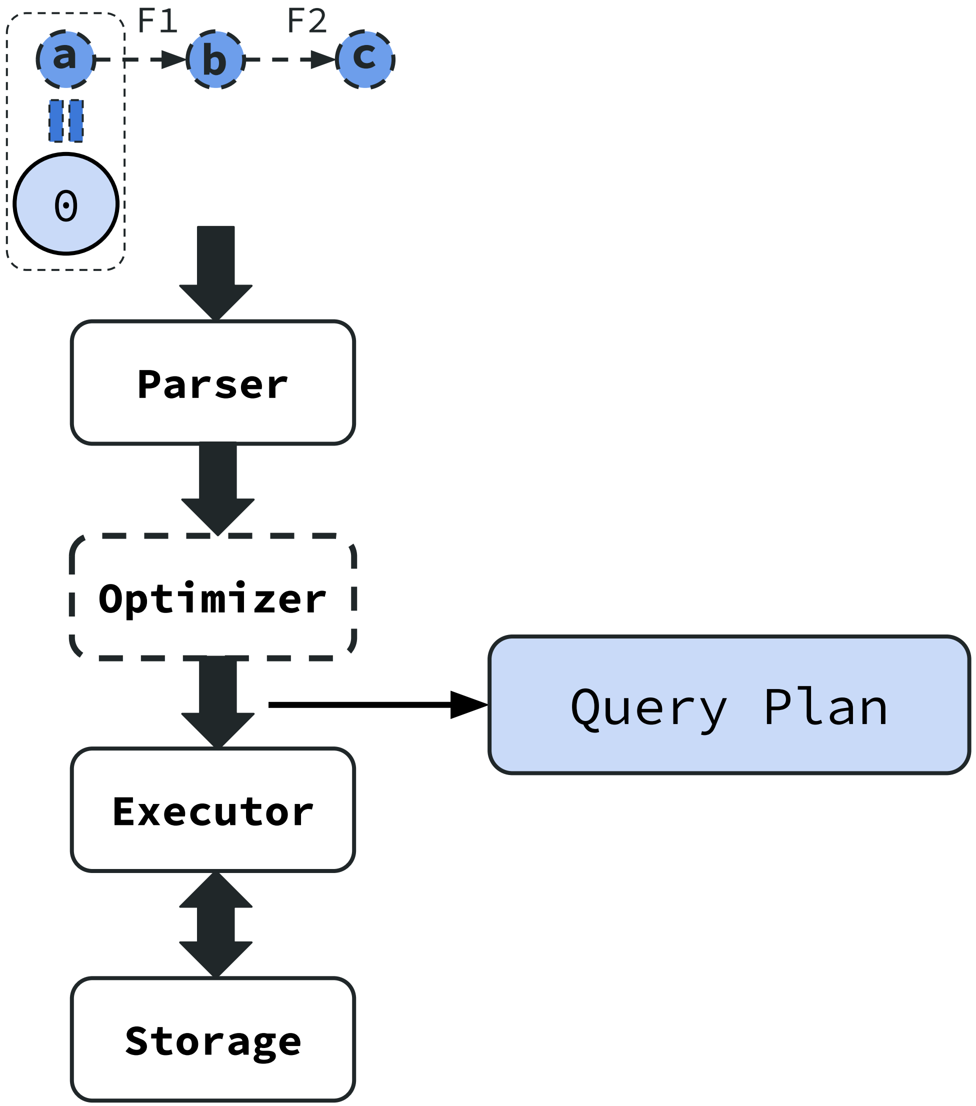
Revisiting DBMS Components (Example)



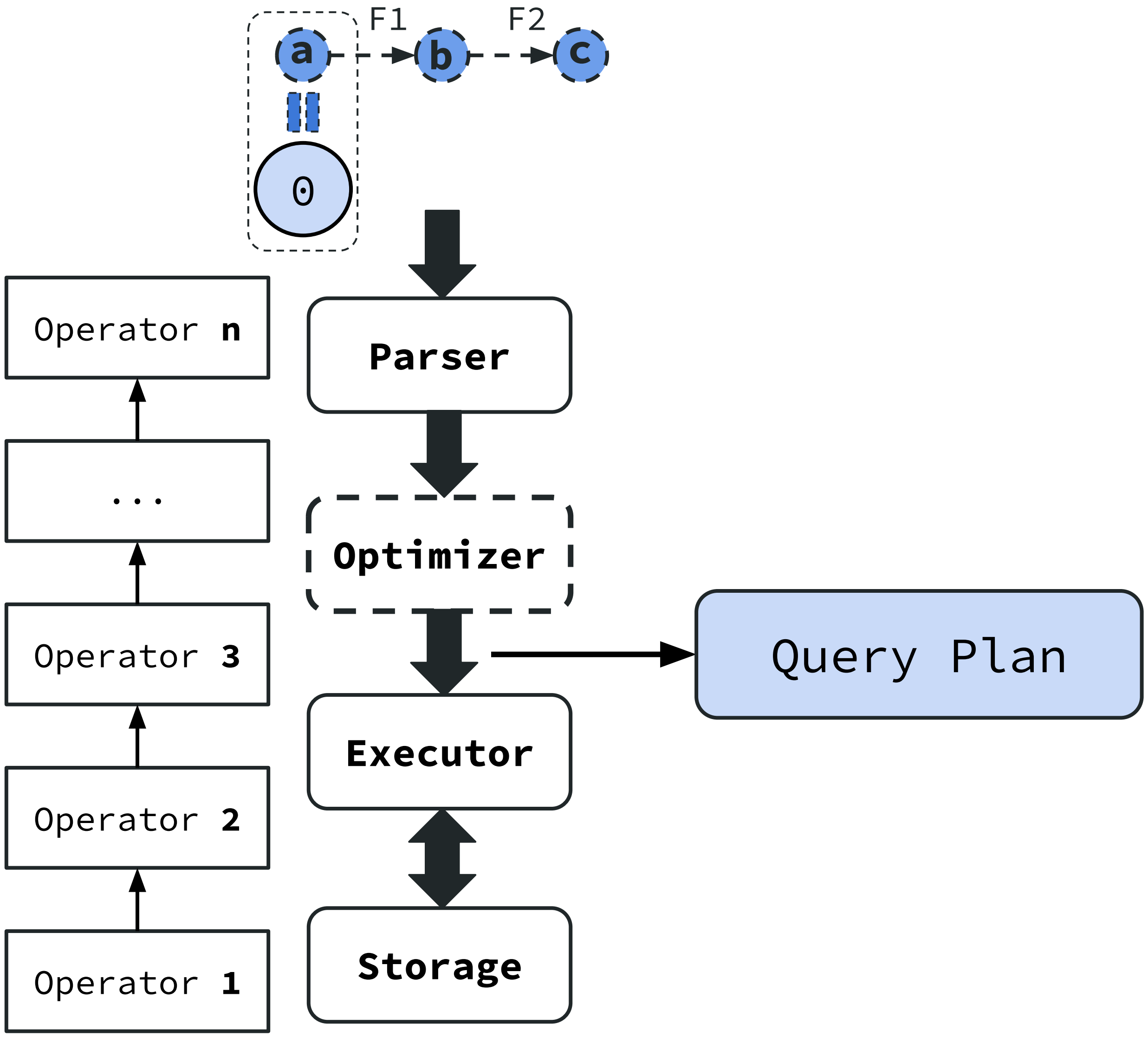
Revisiting DBMS Components (Example)



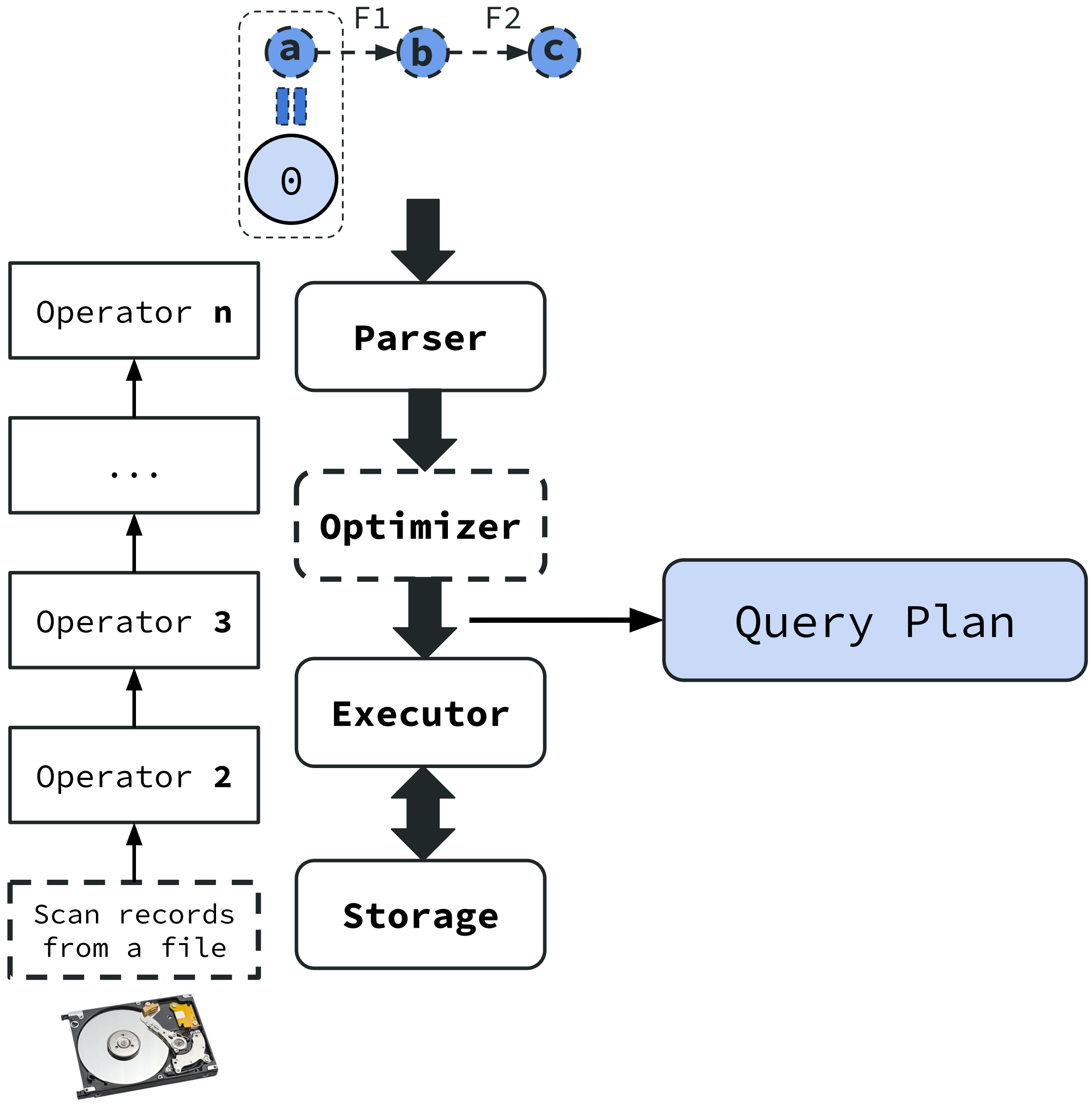
Revisiting DBMS Components (Example)



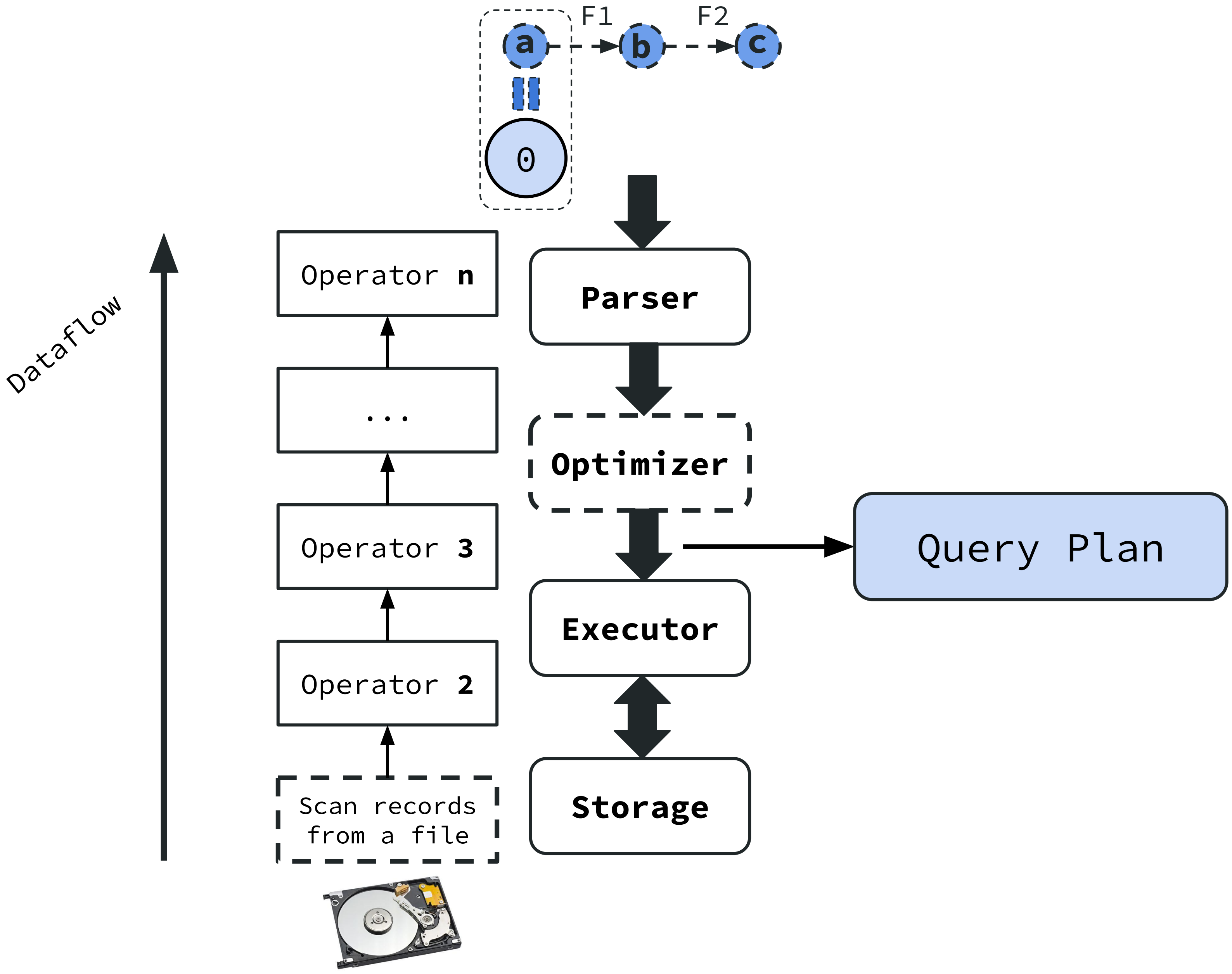
Revisiting DBMS Components (Example)



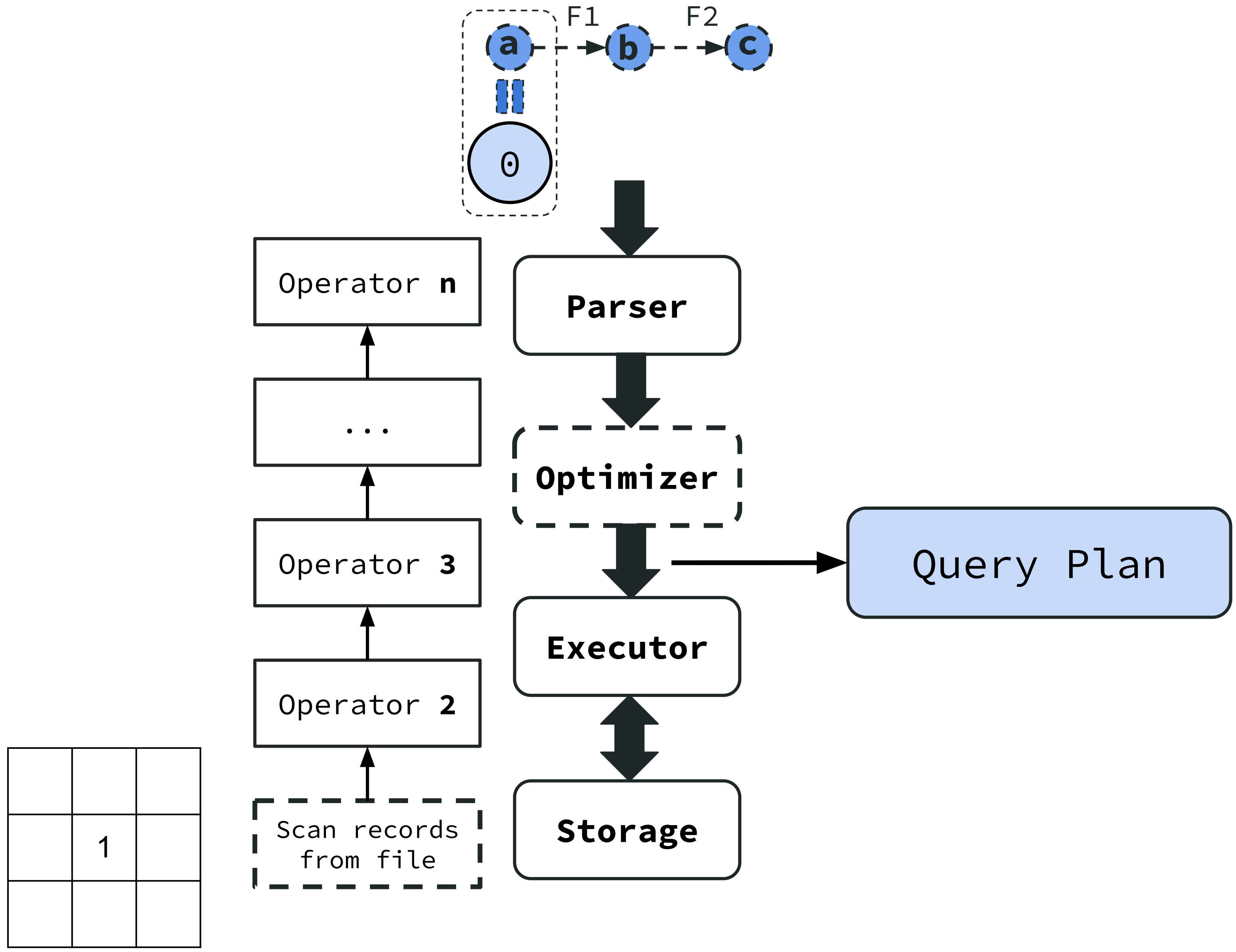
Revisiting DBMS Components (Example)



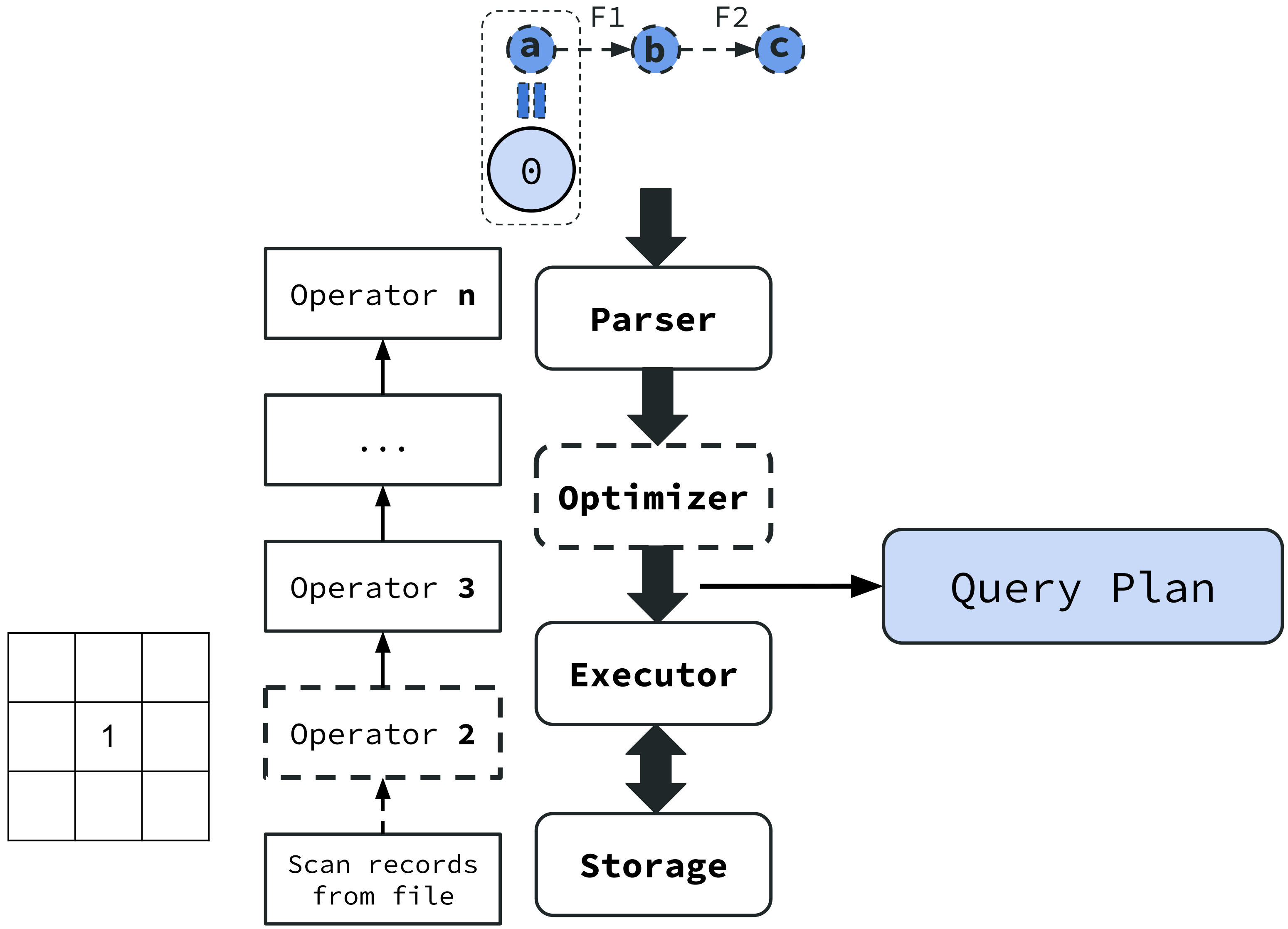
Revisiting DBMS Components (Example)



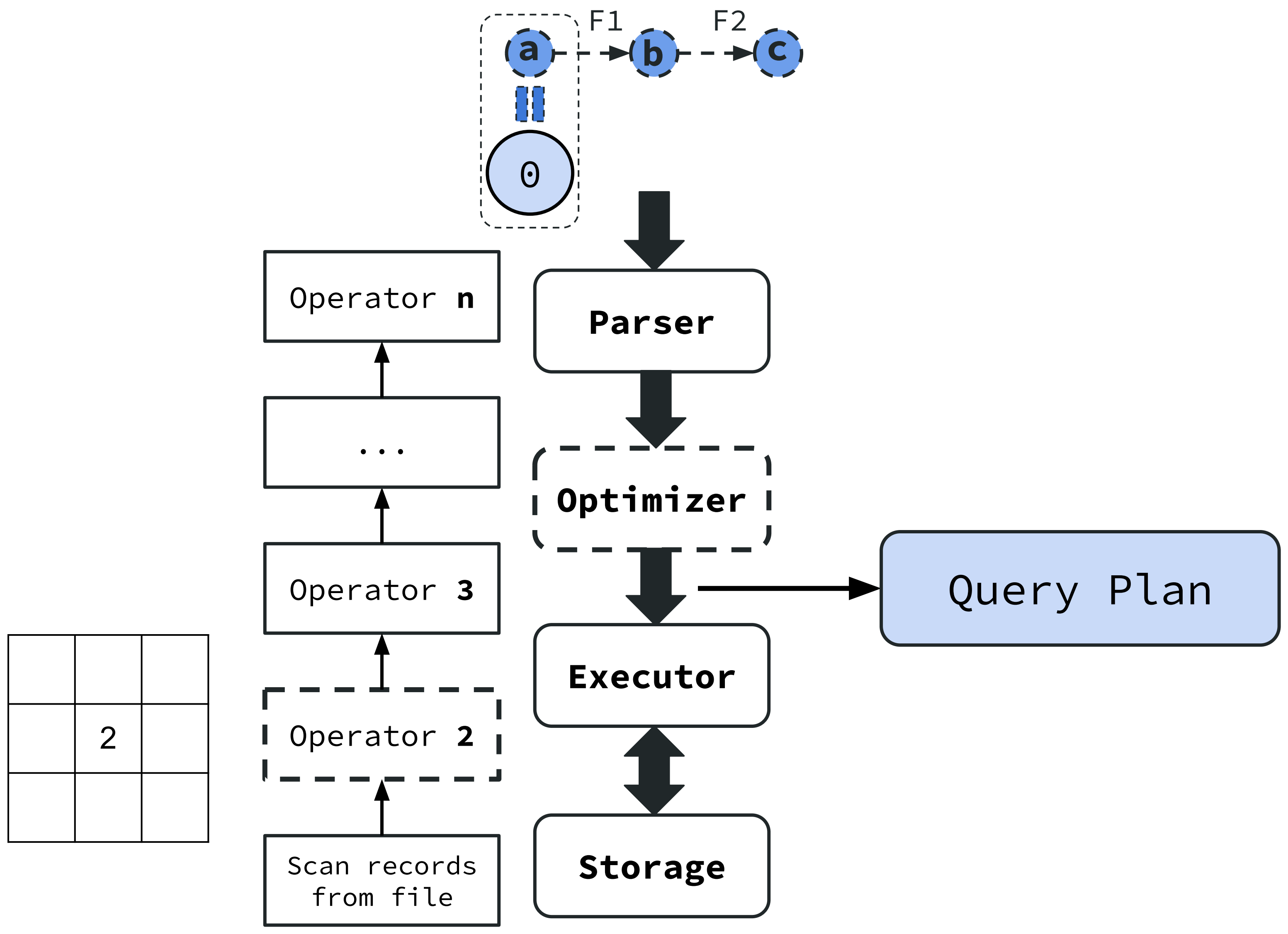
Revisiting DBMS Components (Example)



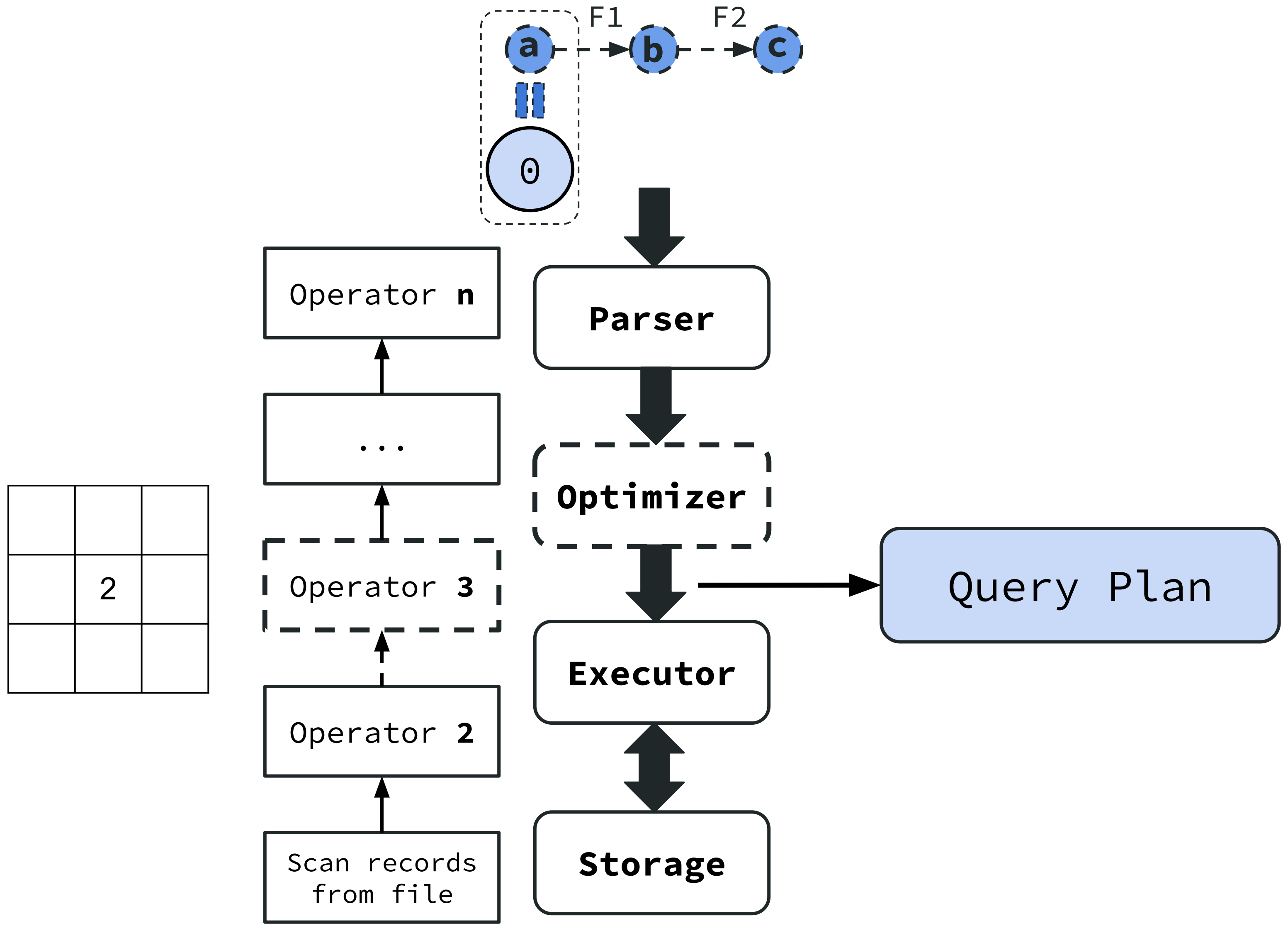
Revisiting DBMS Components (Example)



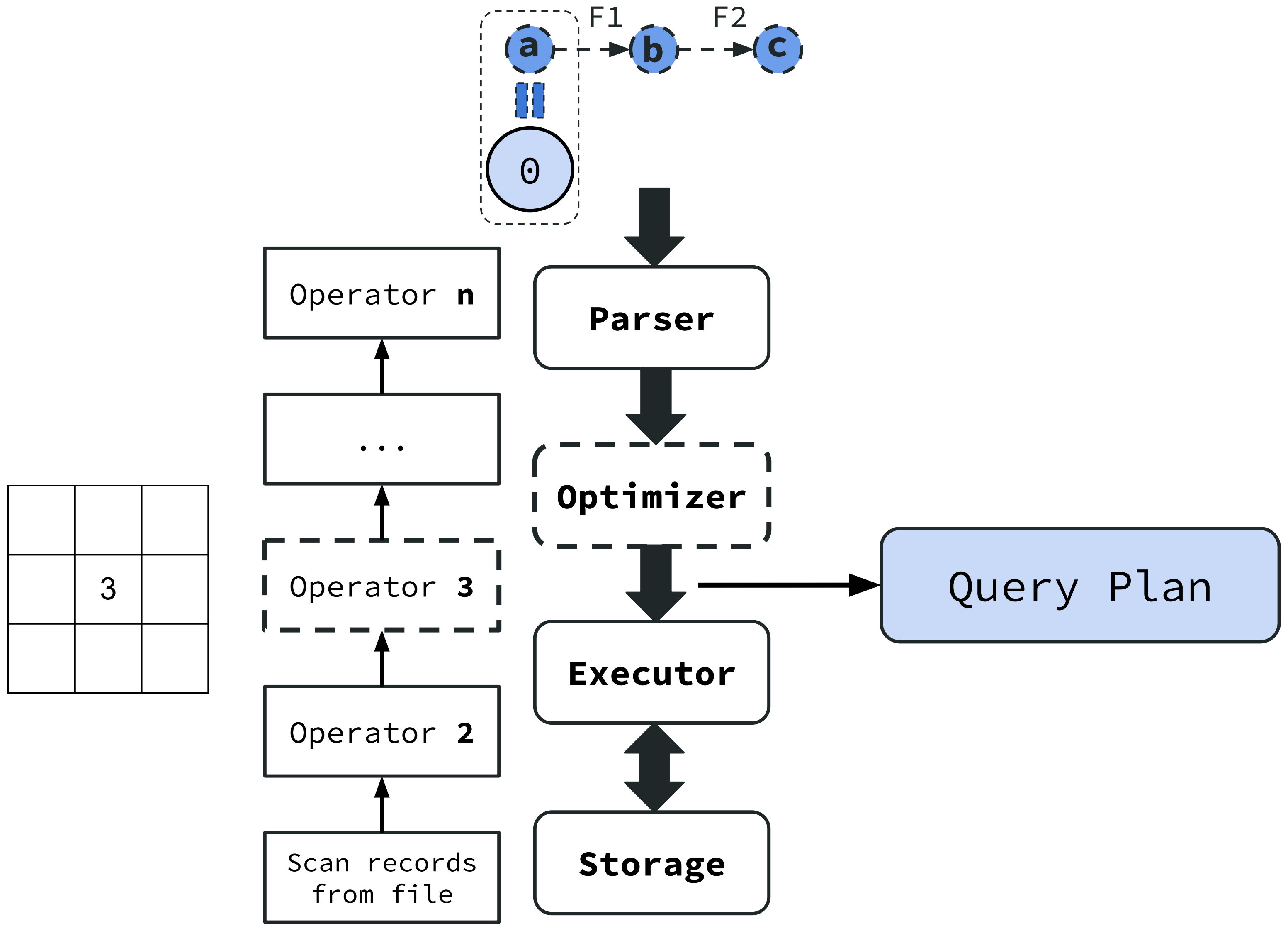
Revisiting DBMS Components (Example)



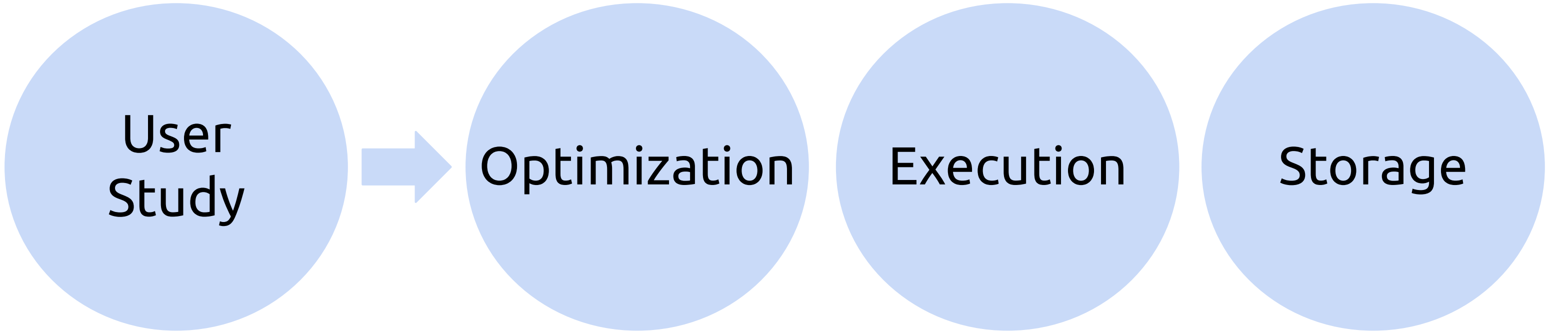
Revisiting DBMS Components (Example)



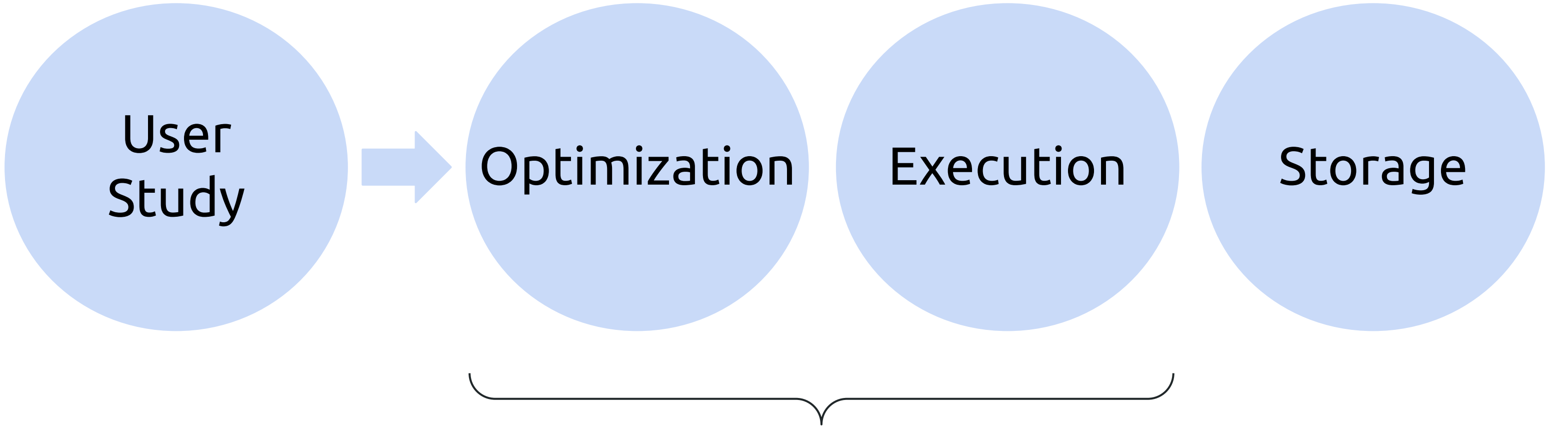
Revisiting DBMS Components (Example)



Research Overview



Research Overview



Novel Join Algorithms

VLDB 2019, TODS 2021

Compressed Representations

VLDB 2021

Query Processing Techniques Overview

Insights

1. Use novel join algorithms to remove unnecessary intermediate results!
2. Use compression to reduce the size of necessary intermediate results!

Query Processing Techniques Overview

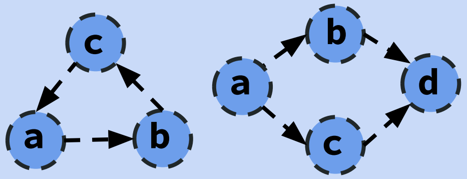
Insights

1. Use novel join algorithms to remove unnecessary intermediate results!

Traditional Joins are *suboptimal!*

Novel Join Algorithms correct the suboptimality

Cyclic Pattern/Join Query



2. Use compression to reduce the size of necessary intermediate results!

Query Processing Techniques Overview

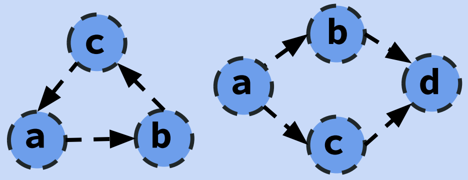
Insights

1. Use novel join algorithms to remove unnecessary intermediate results!

Traditional Joins are suboptimal!

Novel Join Algorithms correct the suboptimality

Cyclic Pattern/Join Query

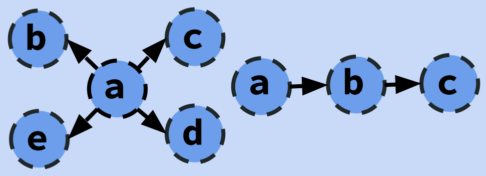


2. Use compression to reduce the size of necessary intermediate results!

Large results part of the final output

Contain a lot of redundancy → compression

Acyclic Pattern/Join Query



Query Processing Techniques Overview

Insights

1. Use novel join algorithms to remove unnecessary intermediate results!
2. Use compression to reduce the size of necessary intermediate results!

Query Processing Techniques Overview

<u>Insights</u>

1. Use novel join algorithms to remove unnecessary intermediate results!
2. Use compression to reduce the size of necessary intermediate results!

<u>Our approach provides</u>:

1. Up to ~10-70x speedups over State-of-the-art!

2. Queries run to completion!

Query Processing Techniques Overview

Insights

1. Use novel join algorithms to remove unnecessary intermediate results!
2. Use compression to reduce the size of necessary intermediate results!

Our approach provides:

1. Up to ~10-70x speedups over State-of-the-art!

2. Queries run to completion!

GraphScope

Alibaba

KÙZU

Outline

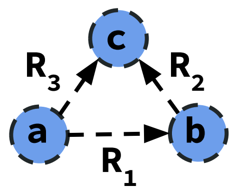
- **Novel Join Algorithms**
  → Worst-case Optimal Joins
- **Compressed Representations**
  → Factorized Representations

Outline

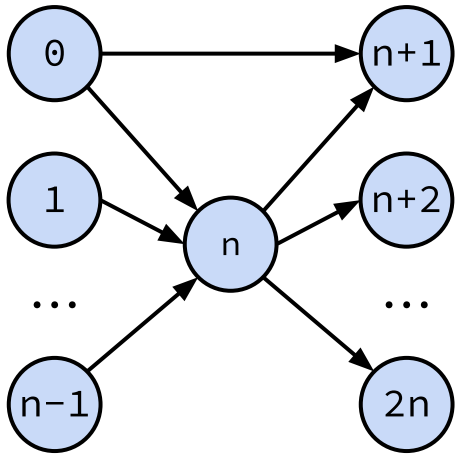
- **Novel Join Algorithms**
  → Worst-case Optimal Joins
- **Compressed Representations**
  → Factorized Representations

Traditional Binary Joins Example

Query

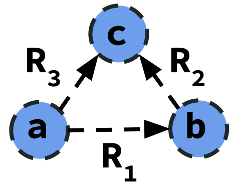


Dataset

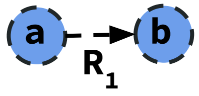


Traditional Binary Joins Example

Query

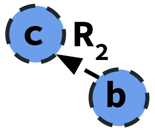


(a, b)
(0, n)
...
(n, 2n)



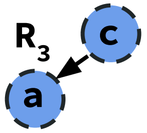
⋈

(b, c)
(0, n)
...
(n, 2n)

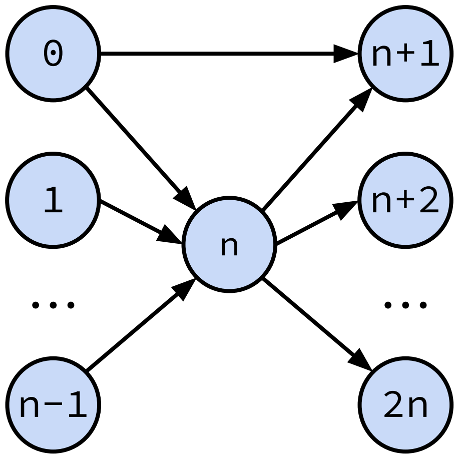


⋈

(a, c)
(0, n)
...
(n, 2n)

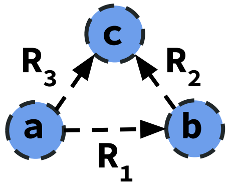


Dataset

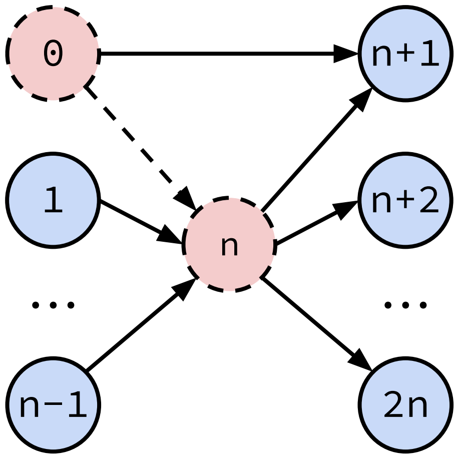


Traditional Binary Joins Example

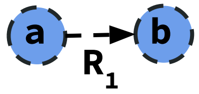
Query

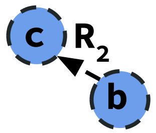


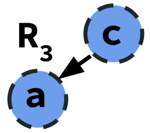
Dataset



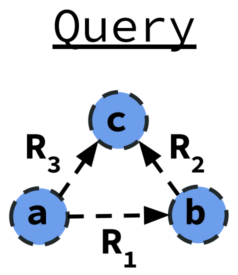
(a, b)	⋈	(b, c)	⋈	(a, c)
(0, n)		(0, n)		(0, n)
...		...		...
(n, 2n)		(n, 2n)		(n, 2n)

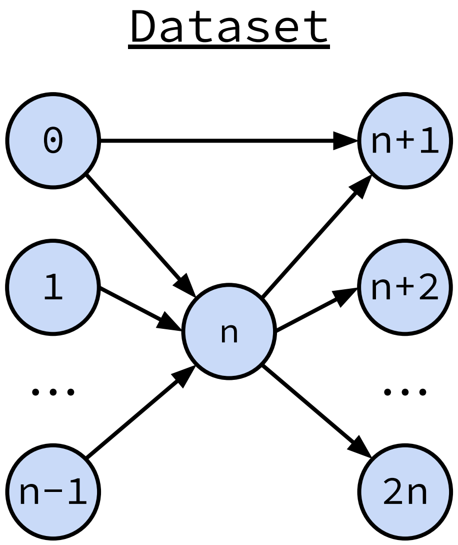




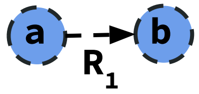


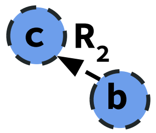
Traditional Binary Joins Example

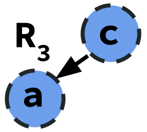




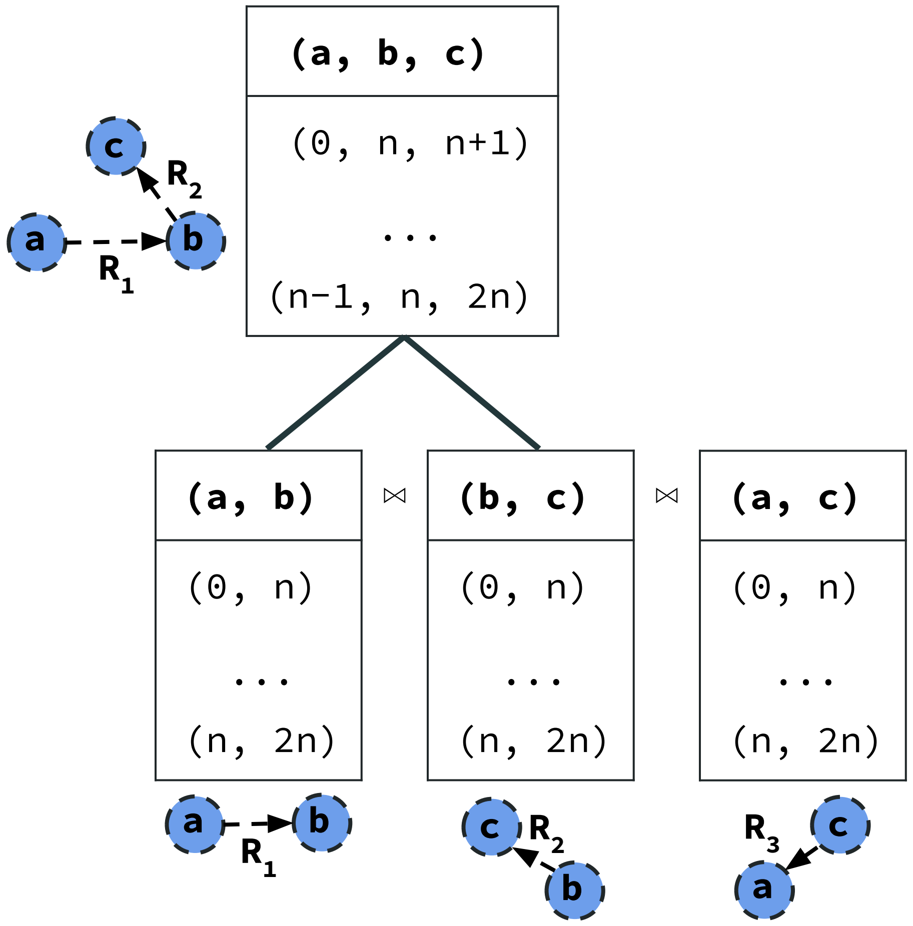
(a, b)	⋈	(b, c)	⋈	(a, c)
(0, n)		(0, n)		(0, n)
...		...		...
(n, 2n)		(n, 2n)		(n, 2n)







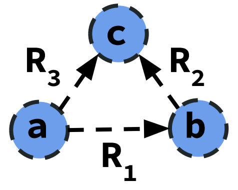
Traditional Binary Joins Example



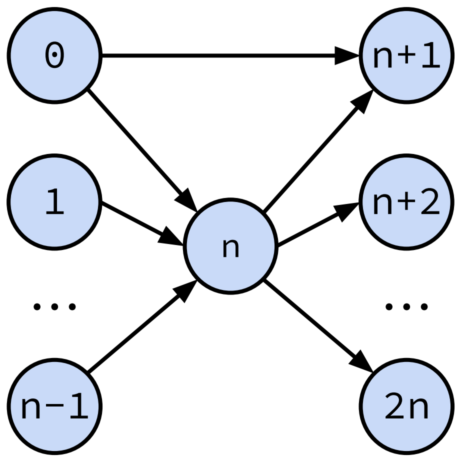
(a, b, c)
(0, n, n+1)
...
(n-1, n, 2n)

(a, b)	⋈	(b, c)	⋈	(a, c)
(0, n)		(0, n)		(0, n)
...		...		...
(n, 2n)		(n, 2n)		(n, 2n)

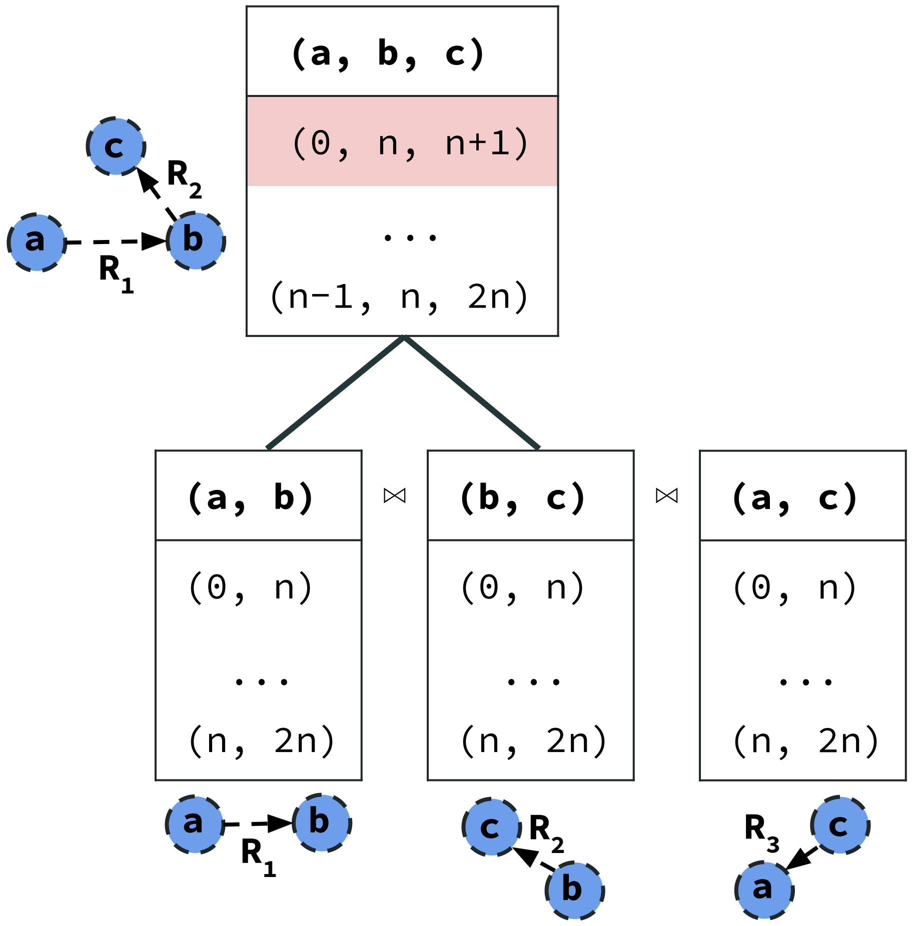
Query



Dataset



Traditional Binary Joins Example



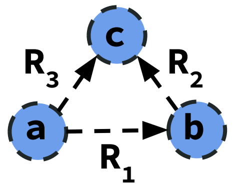
(a, b, c)
(0, n, n+1)
...
(n-1, n, 2n)

(a, b)
(0, n)
...
(n, 2n)

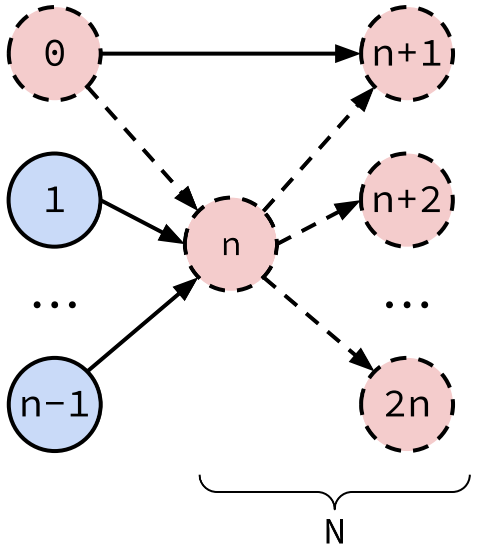
(b, c)
(0, n)
...
(n, 2n)

(a, c)
(0, n)
...
(n, 2n)

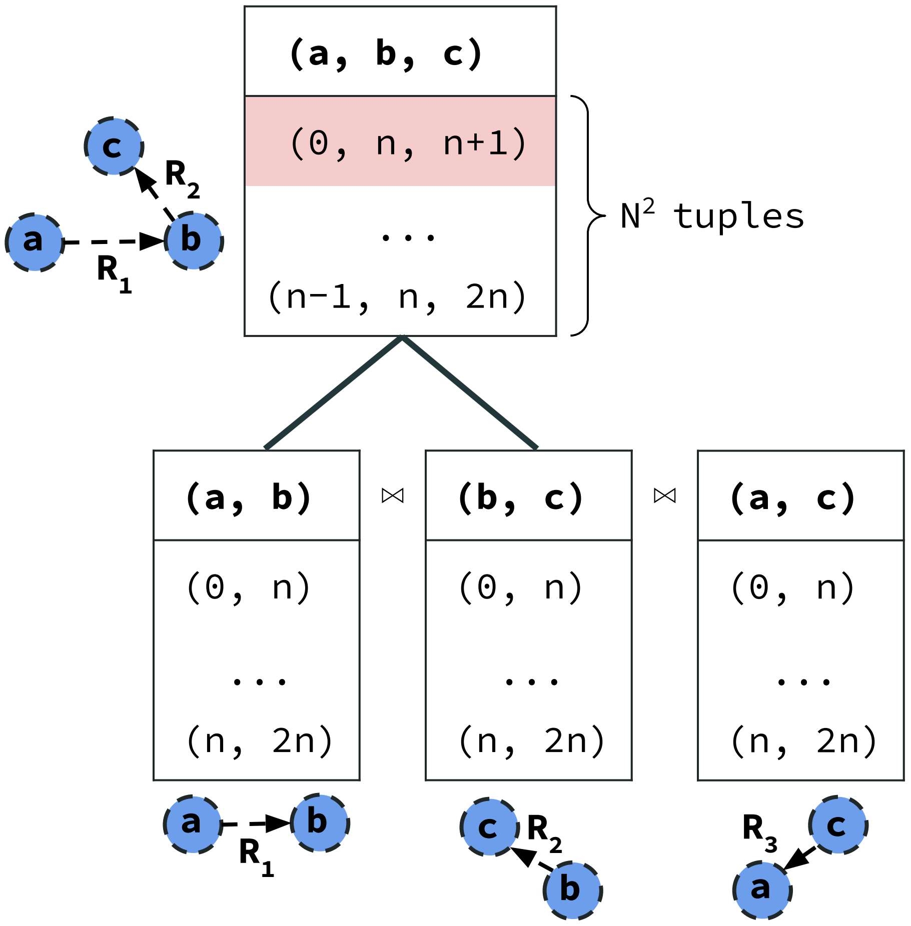
Query



Dataset



Traditional Binary Joins Example



(a, b, c)
(0, n, n+1)
...
(n-1, n, 2n)

(a, b)
(0, n)
...
(n, 2n)

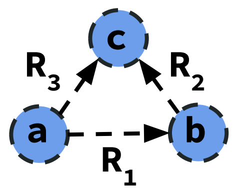
$\bowtie$

(b, c)
(0, n)
...
(n, 2n)

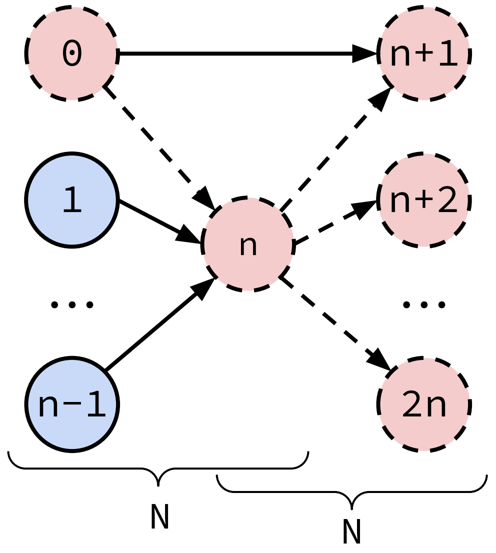
$\bowtie$

(a, c)
(0, n)
...
(n, 2n)

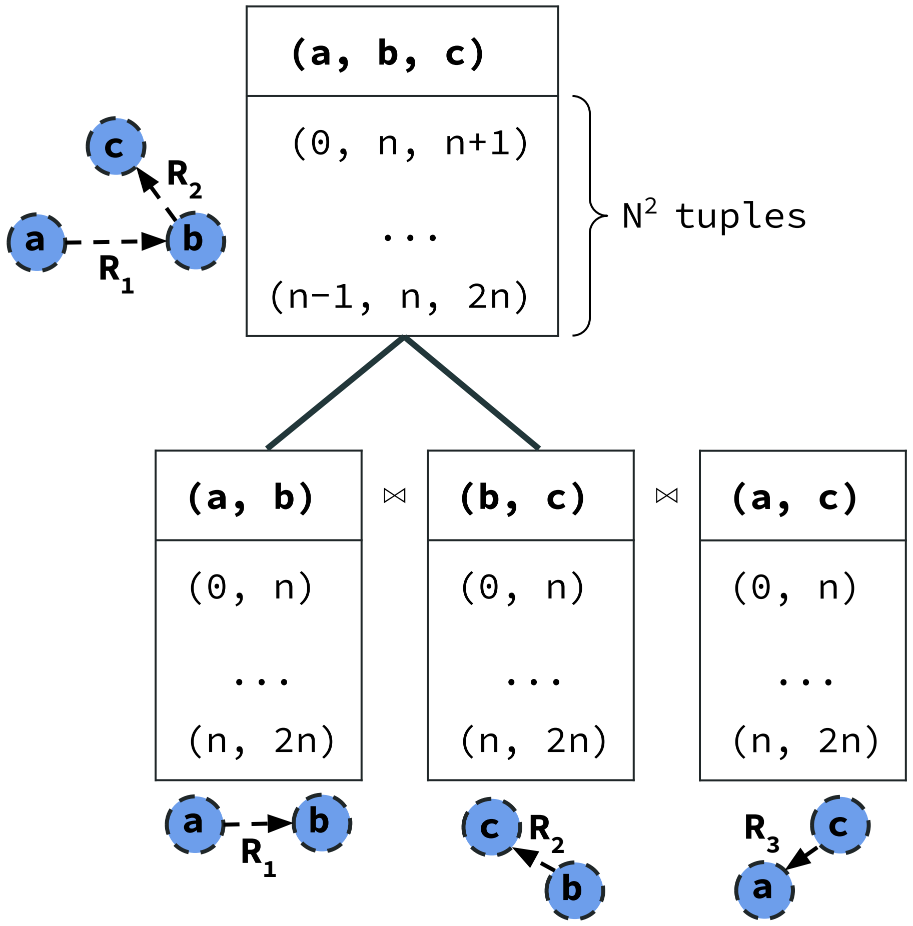
Query



Dataset



Traditional Binary Joins Example



(a, b, c)
(0, n, n+1)
...
(n-1, n, 2n)

N^2 tuples

(a, b)
(0, n)
...
(n, 2n)

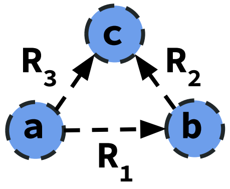
$\bowtie$

(b, c)
(0, n)
...
(n, 2n)

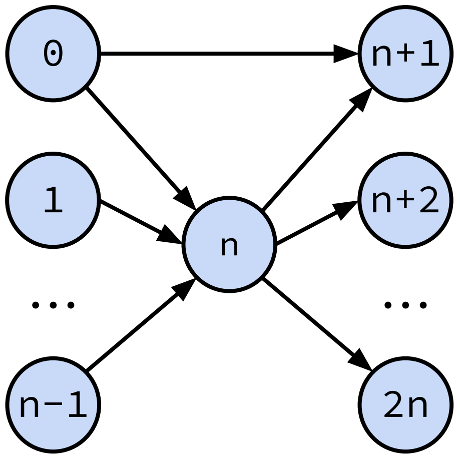
$\bowtie$

(a, c)
(0, n)
...
(n, 2n)

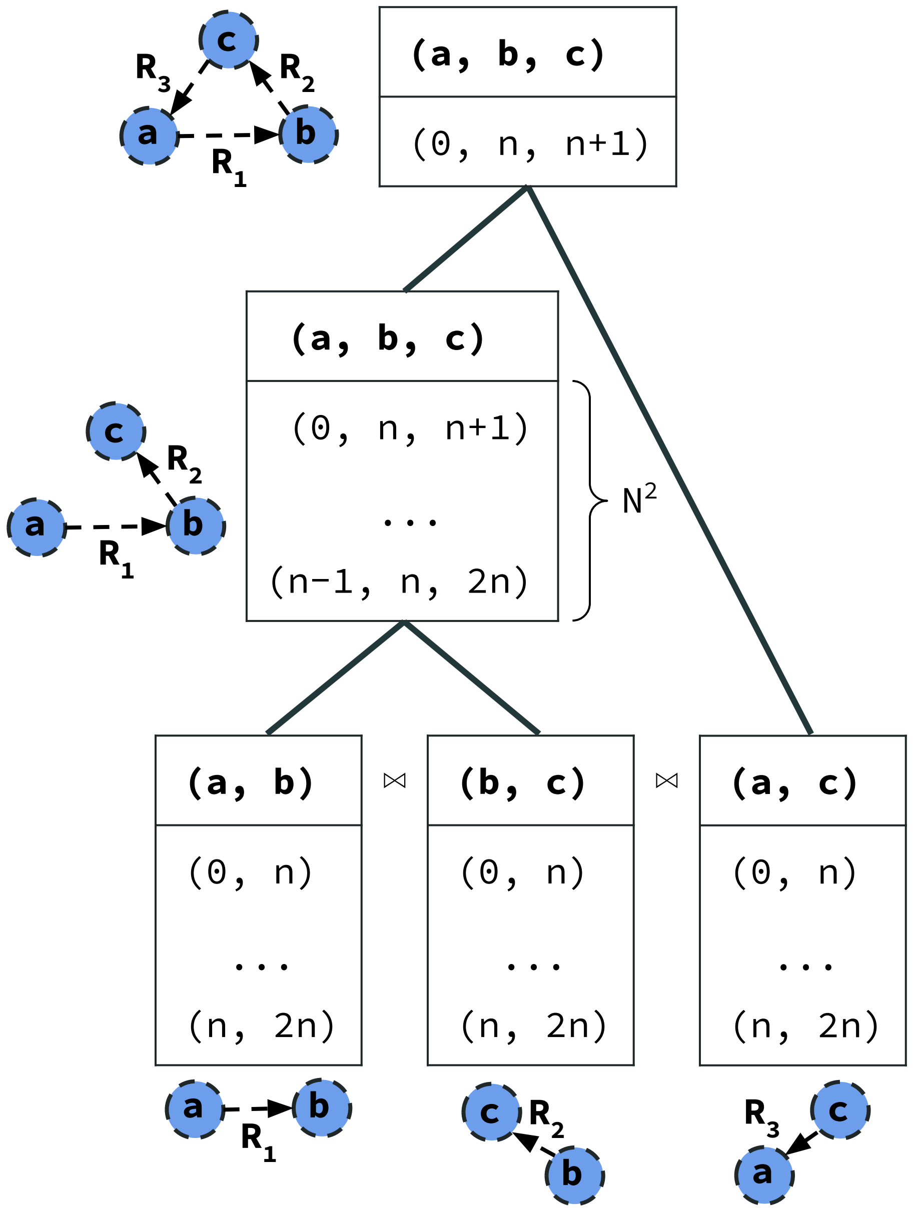
Query



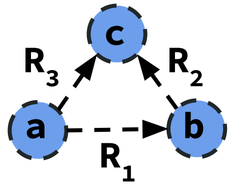
Dataset



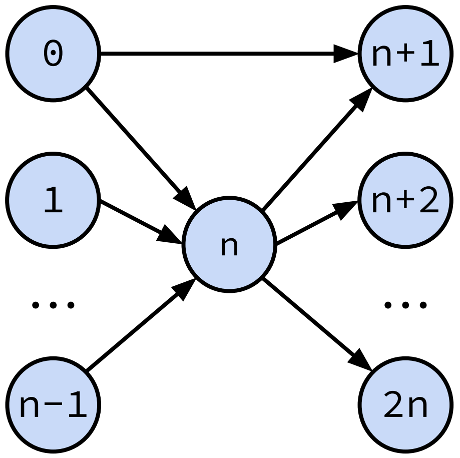
Traditional Binary Joins Example



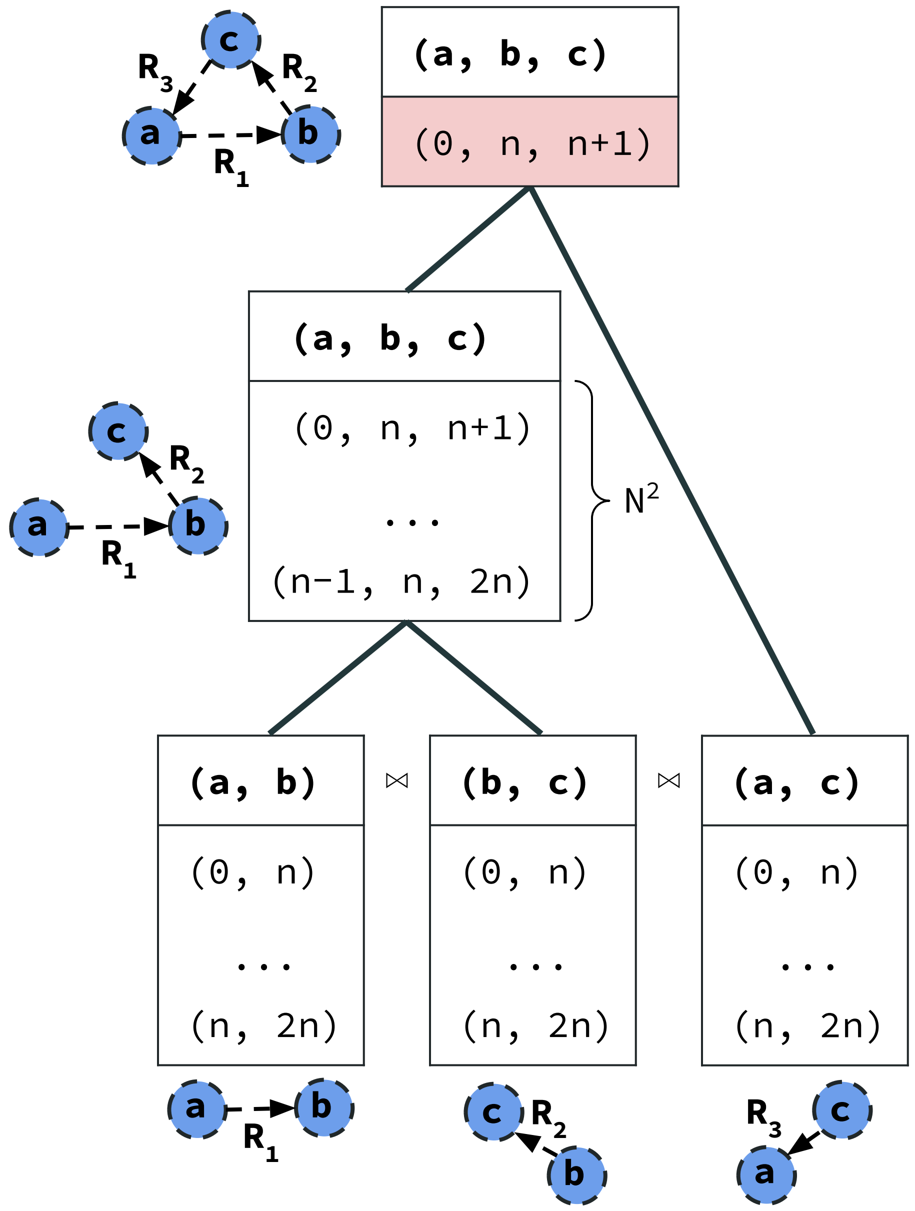
Query



Dataset



Traditional Binary Joins Example



(a, b, c)
(0, n, n+1)

(a, b, c)
(0, n, n+1)
...
(n−1, n, 2n)

(a, b)
(0, n)
...
(n, 2n)

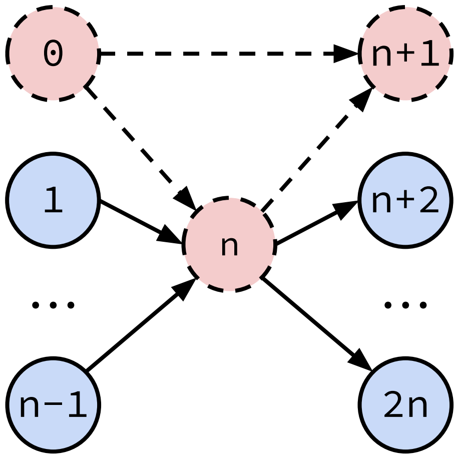
(b, c)
(0, n)
...
(n, 2n)

(a, c)
(0, n)
...
(n, 2n)

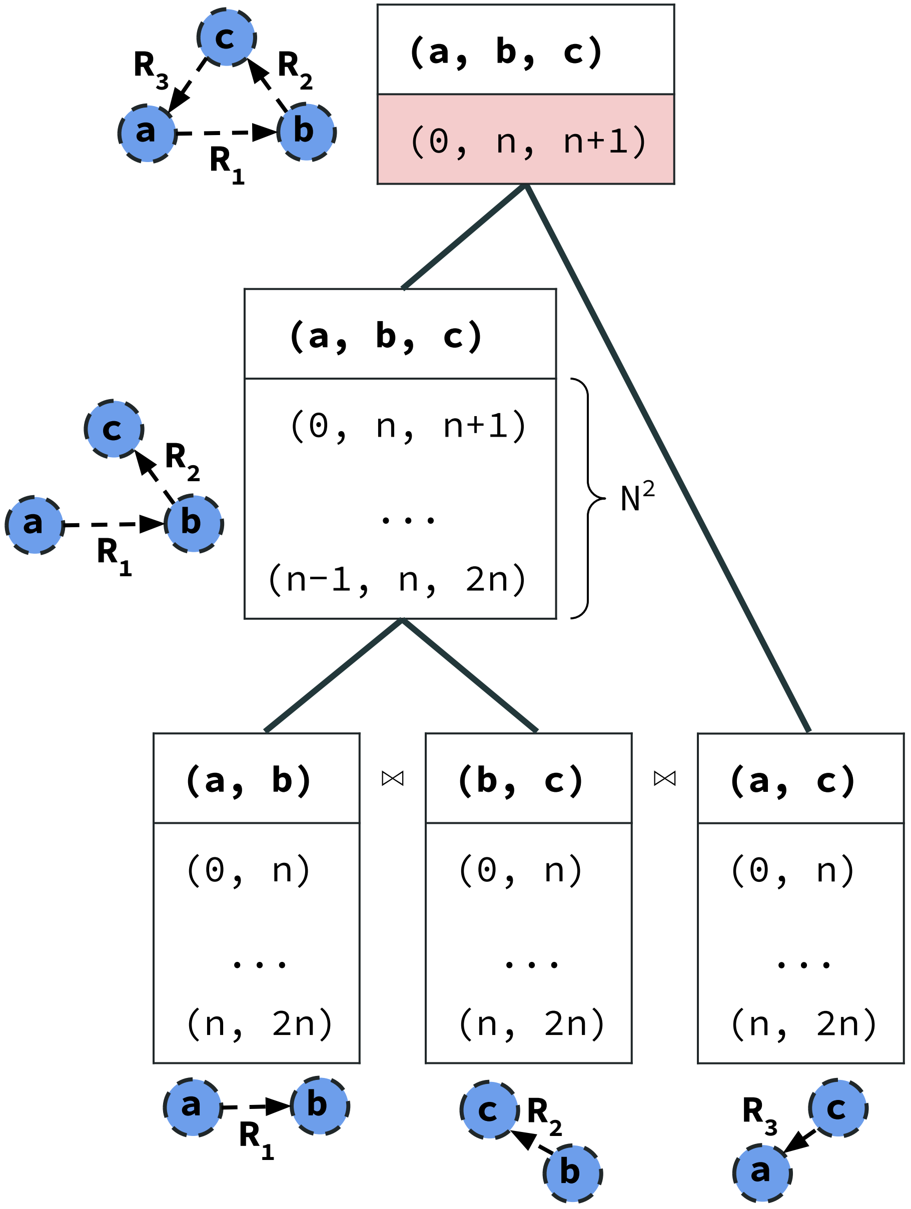
Problem With Binary Join Plans
Prohibitively large # intermediate results for some Q.

For triangles, **AGM bound** is $N^{3/2}$
[FOCS 2008, PODS 2012]

Dataset



Traditional Binary Joins Example

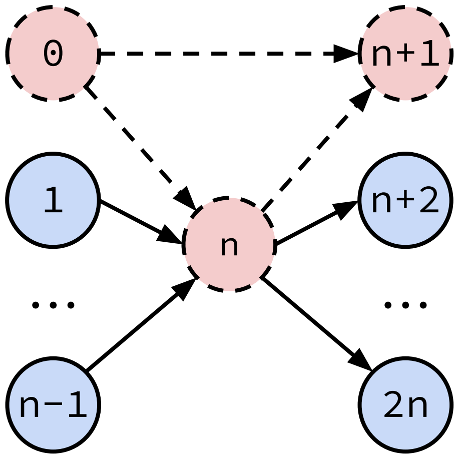


Problem With Binary Join Plans
Prohibitively large # intermediate results for some Q.

For triangles, **AGM bound** is $N^{3/2}$
[FOCS 2008, PODS 2012]

→ ***Worst-case optimal joins correct for the suboptimality*** **(Generic Join)**
SIGMOD Record *[Ngo et al. 2013]*

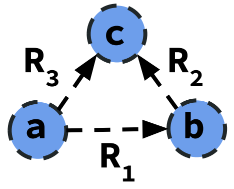
Dataset



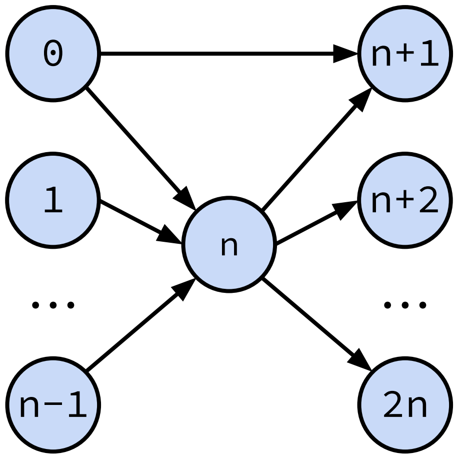
Worst-case Optimal Joins (WCOJs) Example

Worst-case Optimal Joins (WCOJs) Example

Query

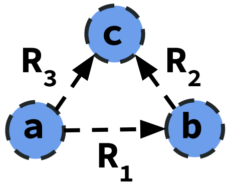


Dataset

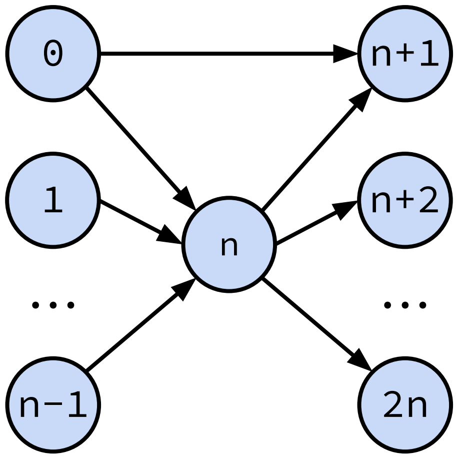


Worst-case Optimal Joins (WCOJs) Example

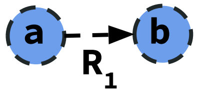
Query

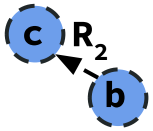


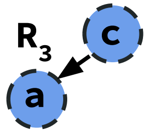
Dataset



(a, b)	⋈	(b, c)	⋈	(a, c)
(0, n)		(0, n)		(0, n)
...		...		...
(n, 2n)		(n, 2n)		(n, 2n)







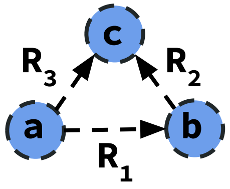
Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

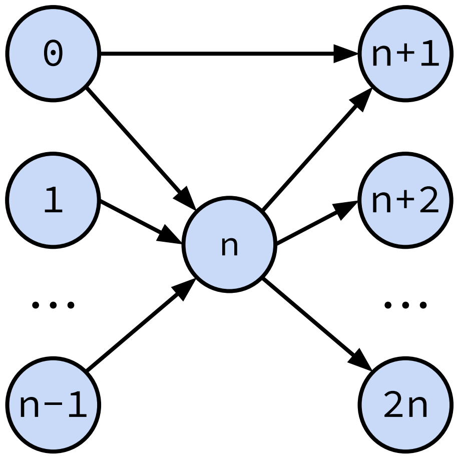
[a,b,c]

1. Find set of a's
2. Given an (a), find b's
3. Given an (a,b), find c's

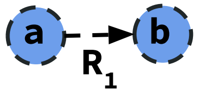
Query

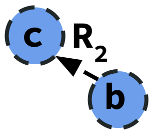


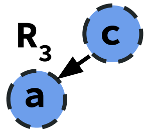
Dataset



(a, b)	⋈	(b, c)	⋈	(a, c)
(0, n)		(0, n)		(0, n)
...		...		...
(n, 2n)		(n, 2n)		(n, 2n)





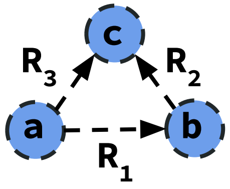


Worst-case Optimal Joins (WCOJs) Example

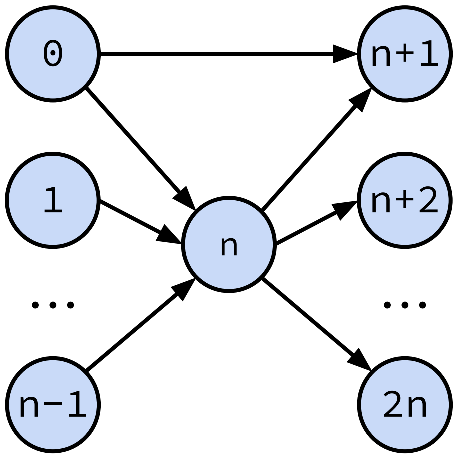
Join Attribute Ordering (JAO)

[a,b,c]

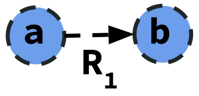
Query

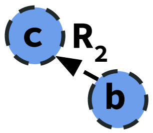


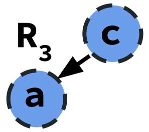
Dataset



(a, b)	⋈	(b, c)	⋈	(a, c)
(0, n)		(0, n)		(0, n)
...		...		...
(n, 2n)		(n, 2n)		(n, 2n)



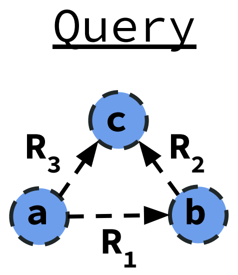


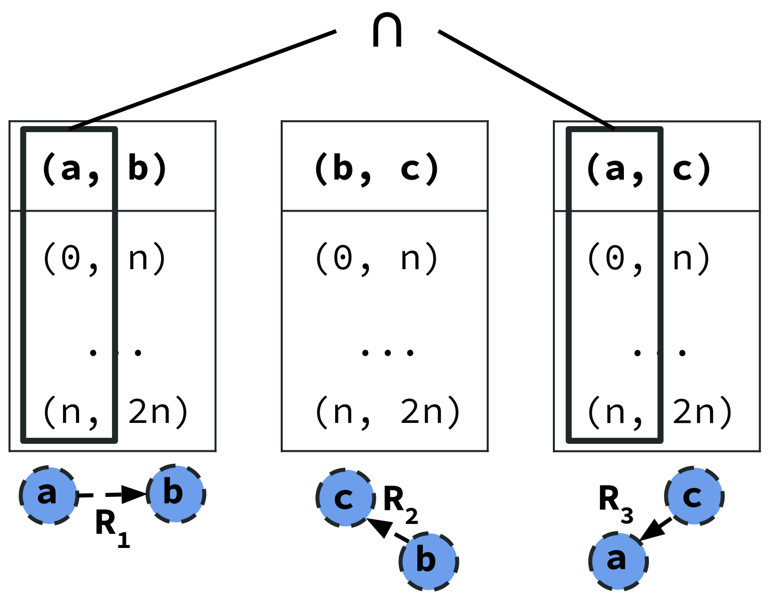


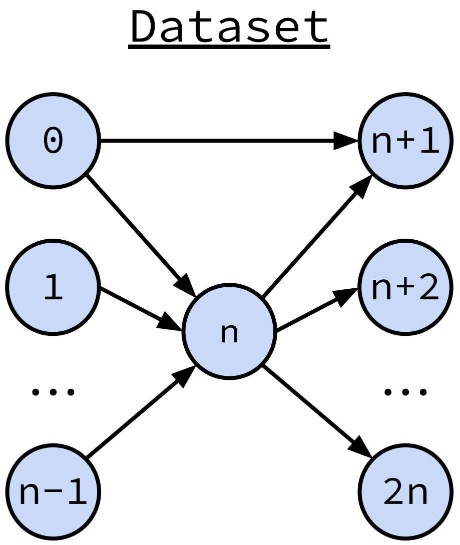
Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]





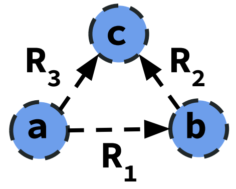


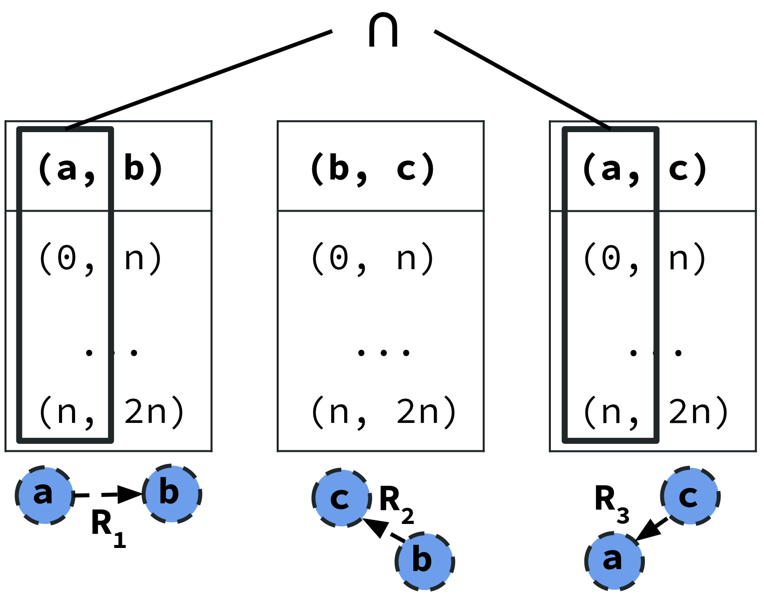
Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]

Query



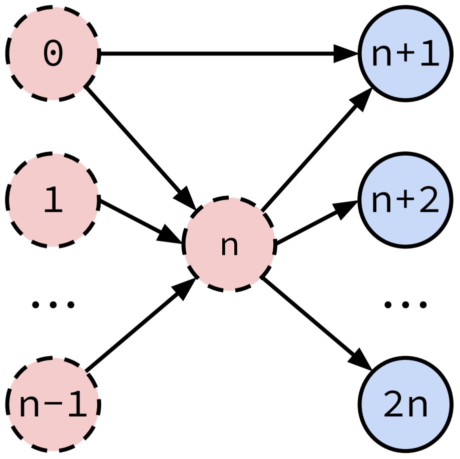


(a, b)
(0, n)
...
(n, 2n)

(b, c)
(0, n)
...
(n, 2n)

(a, c)
(0, n)
...
(n, 2n)

Dataset



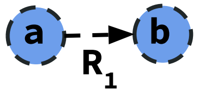
Worst-case Optimal Joins (WCOJs) Example

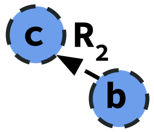
Join Attribute Ordering (JAO)

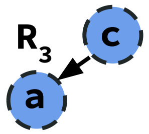
[a,b,c]

(a)
(0)
...
(n)

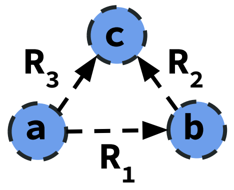
(a, b)	**(b, c)**	**(a, c)**
(0, n)	(0, n)	(0, n)
...	...	...
(n, 2n)	(n, 2n)	(n, 2n)



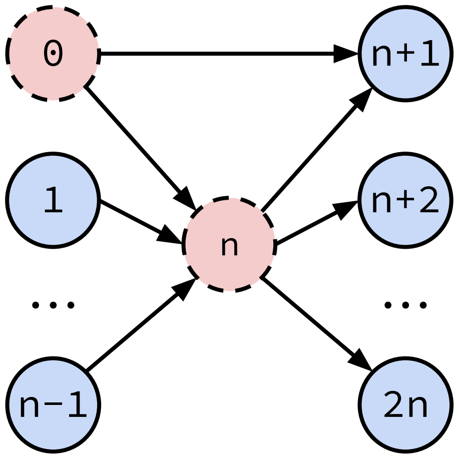




Query



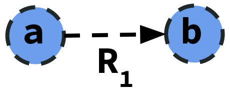
Dataset



Worst-case Optimal Joins (WCOJs) Example

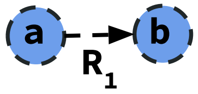
Join Attribute Ordering (JAO)

[a,b,c]

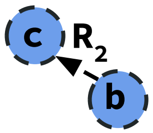


(a)
(0)
...
(n)

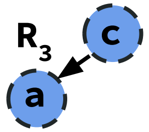
(a, b)
(0, n)
...
(n, 2n)



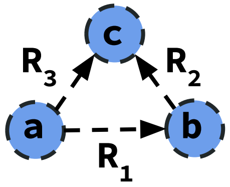
(b, c)
(0, n)
...
(n, 2n)



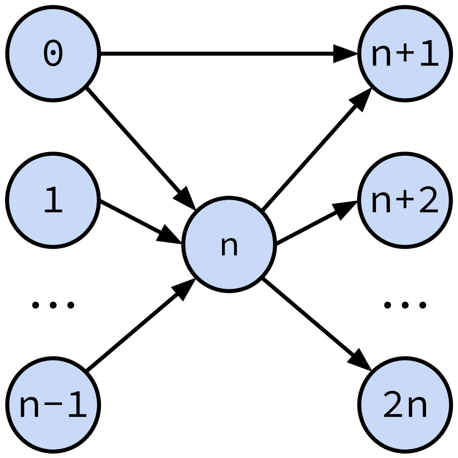
(a, c)
(0, n)
...
(n, 2n)



Query



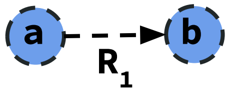
Dataset



Worst-case Optimal Joins (WCOJs) Example

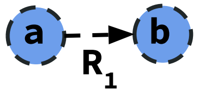
Join Attribute Ordering (JAO)

[a,b,c]

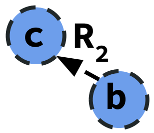


(a)
(0)
...
(n)

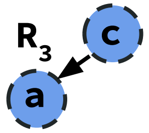
(a, b)
(0, n)
...
(n, 2n)



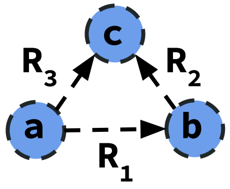
(b, c)
(0, n)
...
(n, 2n)



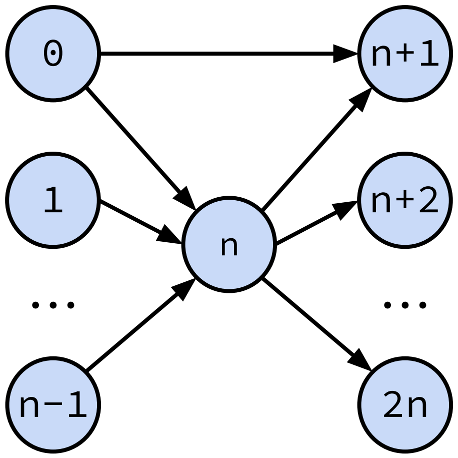
(a, c)
(0, n)
...
(n, 2n)



Query



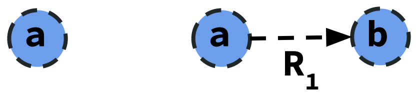
Dataset



Worst-case Optimal Joins (WCOJs) Example

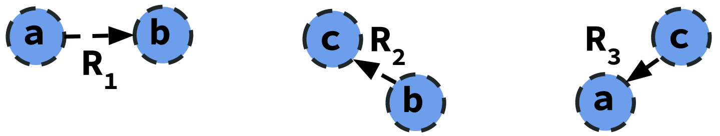
Join Attribute Ordering (JAO)

[a,b,c]

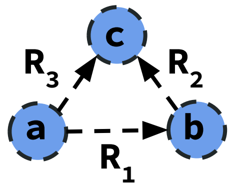


(a)
(0)
...
(n)

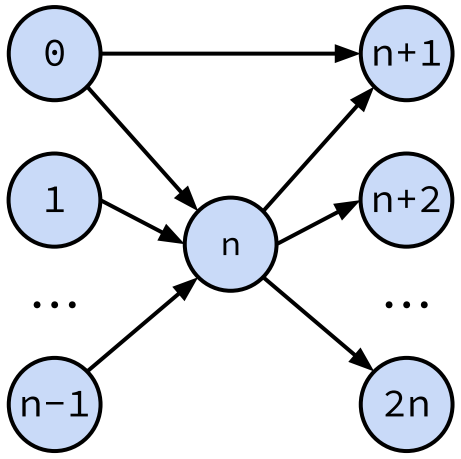
(a, b)	**(b, c)**	**(a, c)**
(0, n)	(0, n)	(0, n)
...	...	...
(n, 2n)	(n, 2n)	(n, 2n)



Query



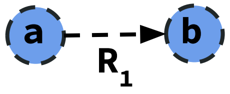
Dataset



Worst-case Optimal Joins (WCOJs) Example

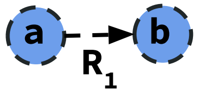
Join Attribute Ordering (JAO)

[a,b,c]

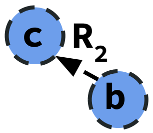


(a)
🟥 (0)
...
(n)

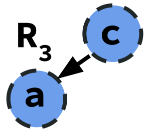
(a,	b)
(0,	n)
...	
(n,	2n)



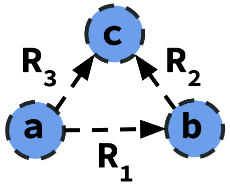
(b,	c)
(0,	n)
...	
(n,	2n)



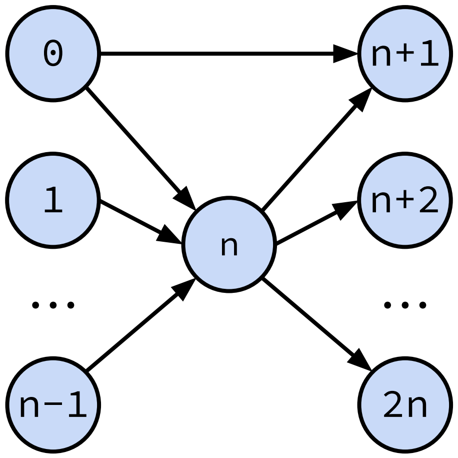
(a, c)
(0, n)
...
(n, 2n)



Query



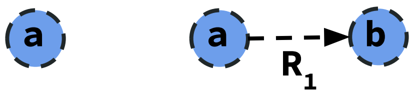
Dataset



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

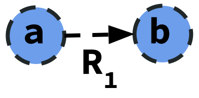
[a,b,c]



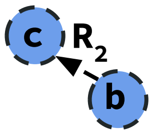
(a)
(0)
...
(n)

∩

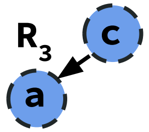
(a, b)
(0, n)
...
(n, 2n)



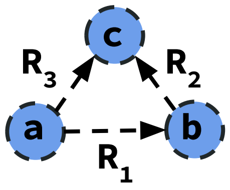
(b, c)
(0, n)
...
(n, 2n)



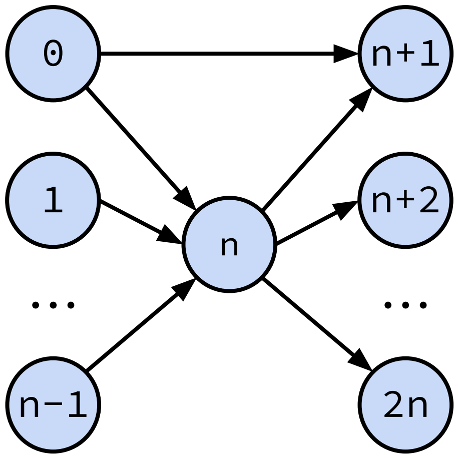
(a, c)
(0, n)
...
(n, 2n)



Query



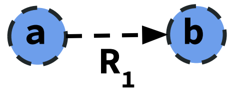
Dataset



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

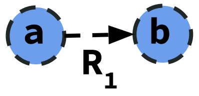
[a,b,c]



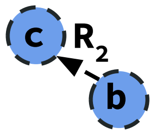
(a)
(0)
...
(n)

∩

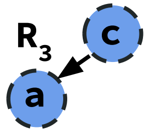
(a, b)
(0, n)
...
(n, 2n)



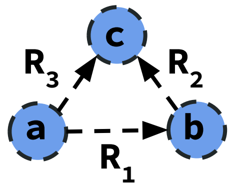
(b, c)
(0, n)
...
(n, 2n)



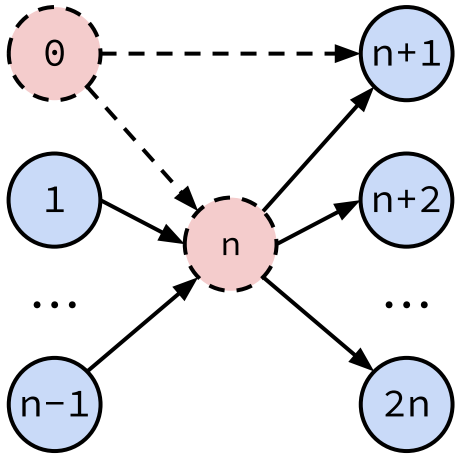
(a, c)
(0, n)
...
(n, 2n)



Query



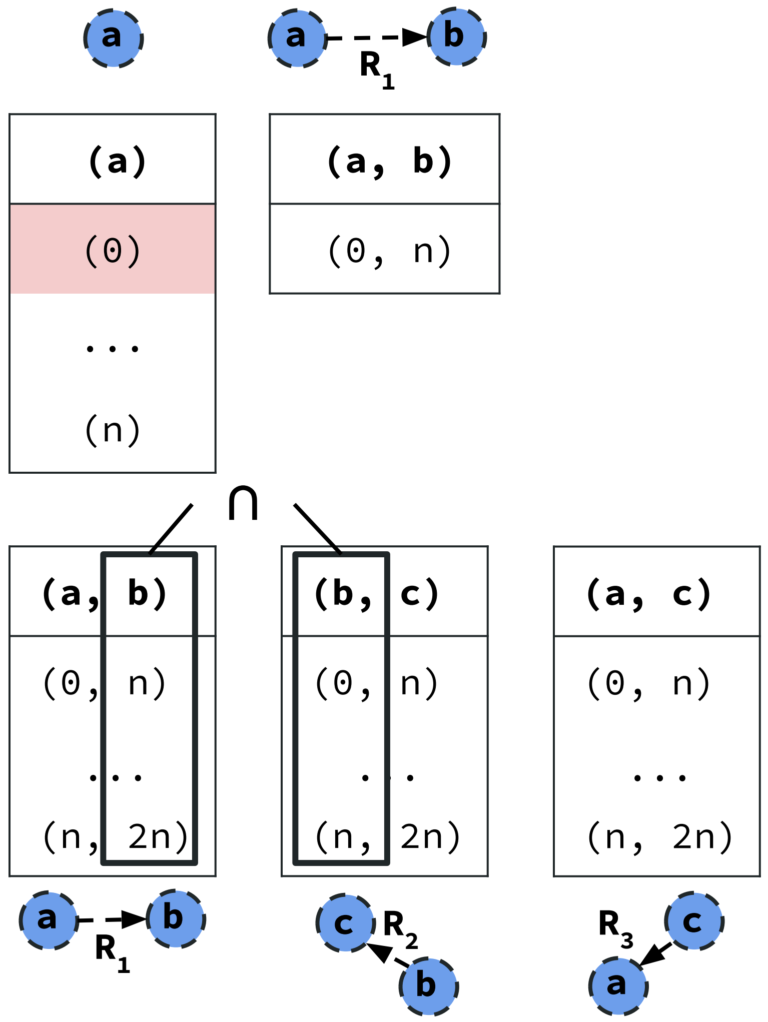
Dataset



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]



(a)
(0)
...
(n)

(a, b)
(0, n)

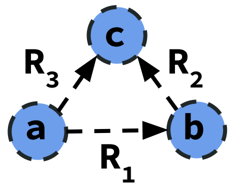
∩

(a, b)
(0, n)
...
(n, 2n)

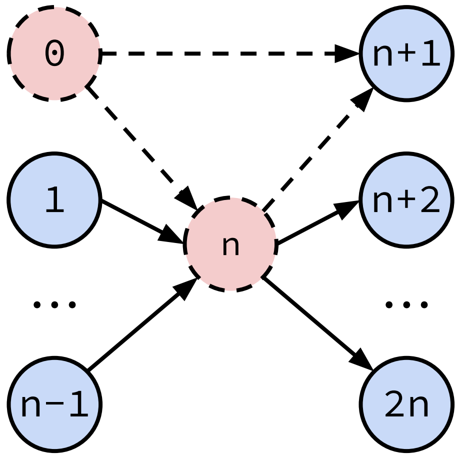
(b, c)
(0, n)
...
(n, 2n)

(a, c)
(0, n)
...
(n, 2n)

Query



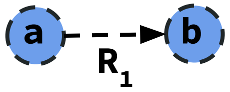
Dataset



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

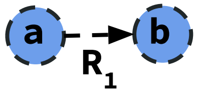
[a,b,c]



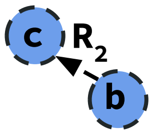
(a)
(0)
...
(n)

(a, b)
(0, n)

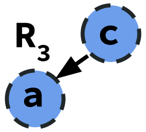
(a, b)
(0, n)
...
(n, 2n)



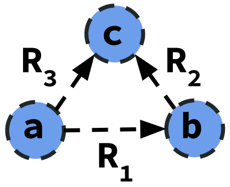
(b, c)
(0, n)
...
(n, 2n)



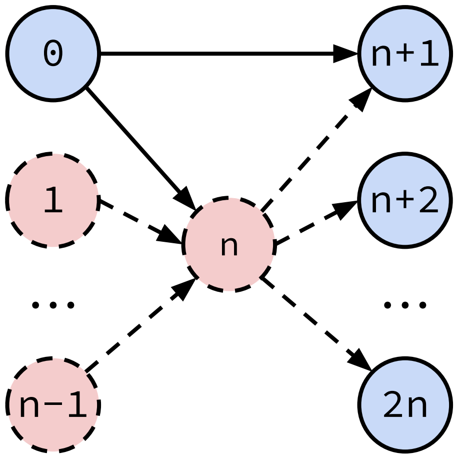
(a, c)
(0, n)
...
(n, 2n)



Query



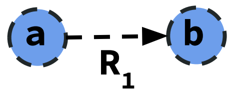
Dataset



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

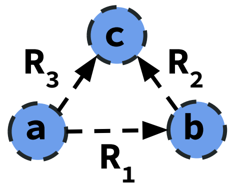
[a,b,c]



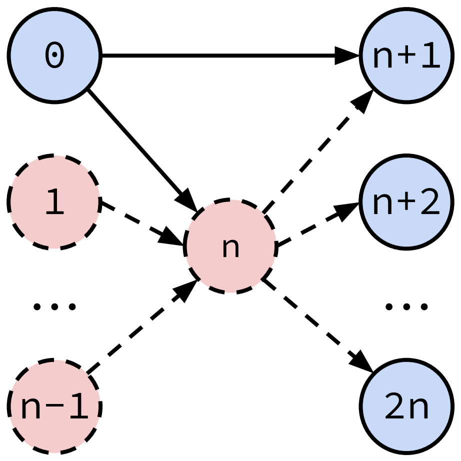
(a)
(0)
...
(n)

(a, b)
(0, n)
(…, n)

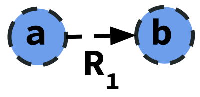
Query



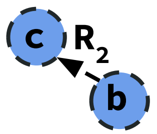
Dataset



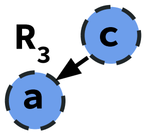
(a, b)
(0, n)
...
(n, 2n)



(b, c)
(0, n)
...
(n, 2n)



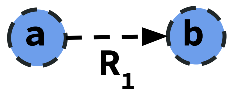
(a, c)
(0, n)
...
(n, 2n)



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]

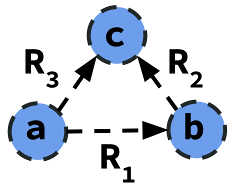


(a)
(0)
...
(n)

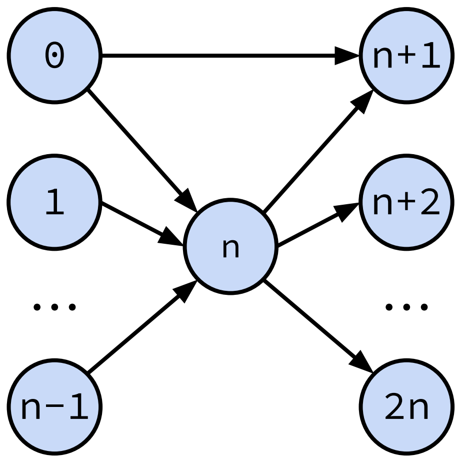
(a, b)
(0, n)
(…, n)

N

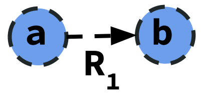
Query



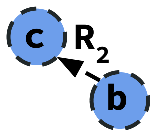
Dataset



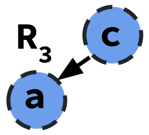
(a, b)
(0, n)
...
(n, 2n)



(b, c)
(0, n)
...
(n, 2n)



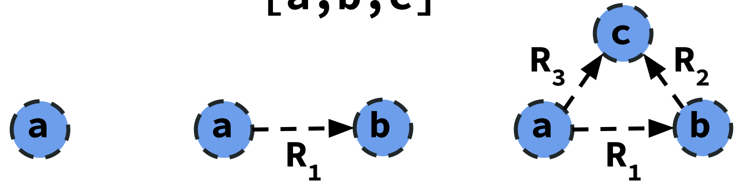
(a, c)
(0, n)
...
(n, 2n)



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]

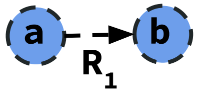


(a)
(0)
...
(n)

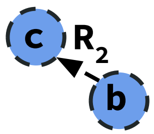
(a, b)
(0, n)
(…, n)

N

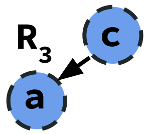
(a, b)
(0, n)
...
(n, 2n)



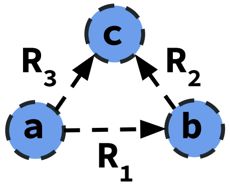
(b, c)
(0, n)
...
(n, 2n)



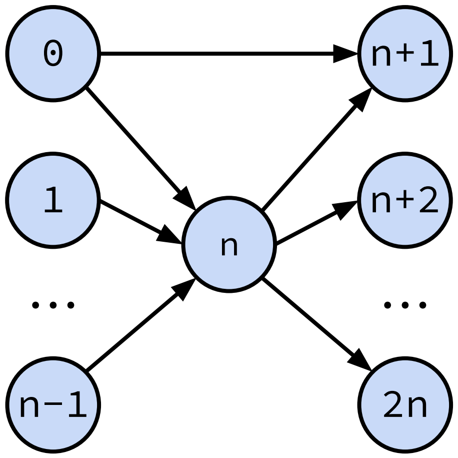
(a, c)
(0, n)
...
(n, 2n)



<u>Query</u>



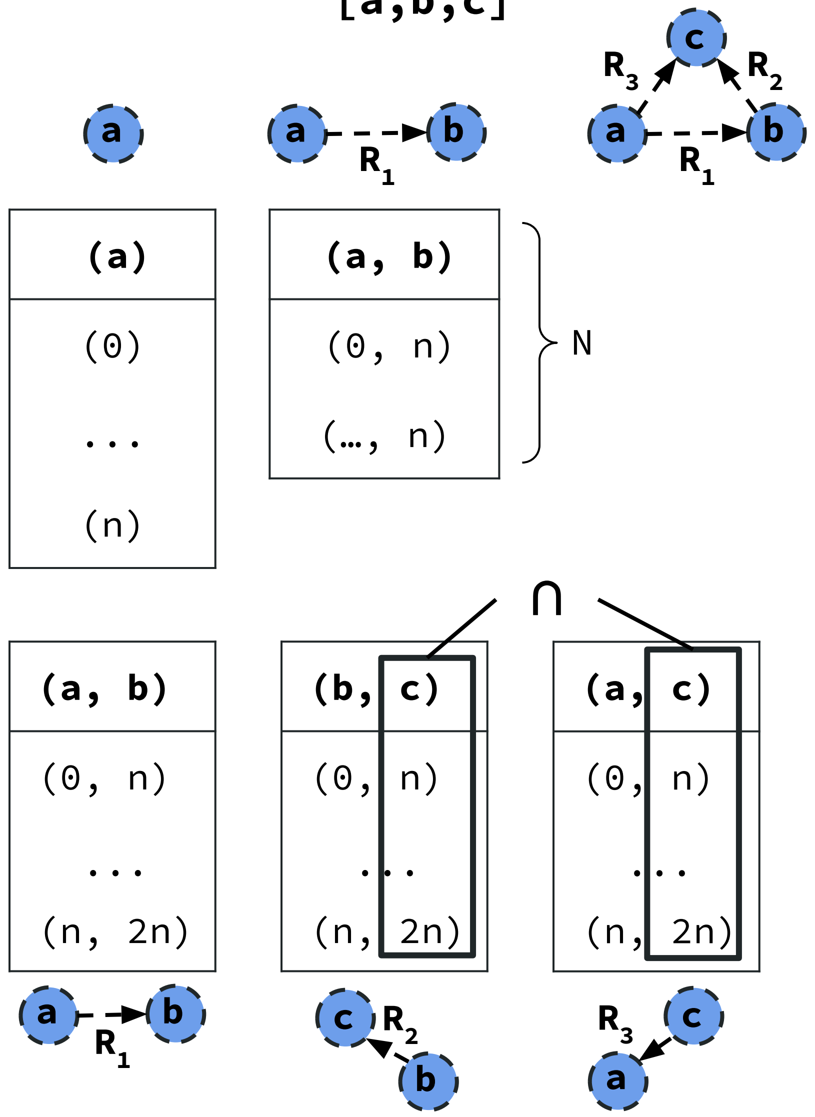
<u>Dataset</u>



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

`[a,b,c]`



(a)
(0)
...
(n)

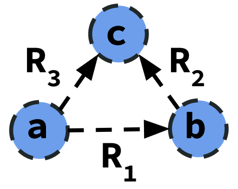
(a, b)
(0, n)
(…, n)

(a, b)
(0, n)
...
(n, 2n)

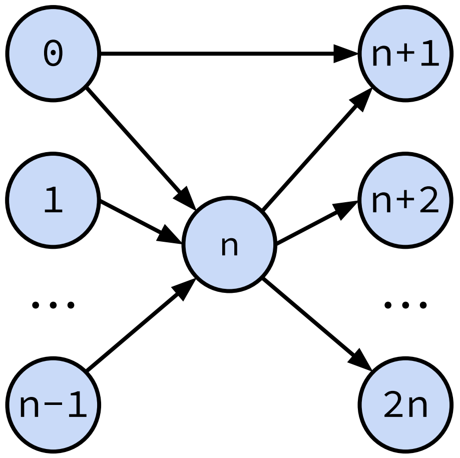
(b, c)
(0, n)
...
(n, 2n)

(a, c)
(0, n)
...
(n, 2n)

Query



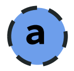
Dataset

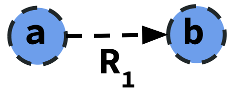


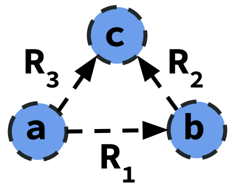
Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]





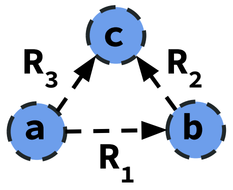


(a)
(0)
...
(n)

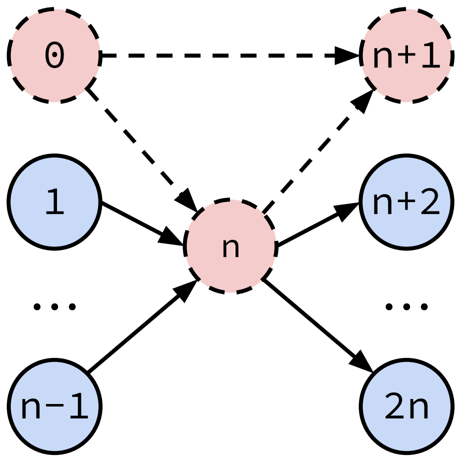
(a, b)
(0, n)
(…, n)

(a, b, c)
(0, n, n+1)

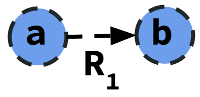
Query



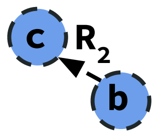
Dataset



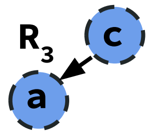
(a, b)
(0, n)
...
(n, 2n)



(b, c)
(0, n)
...
(n, 2n)



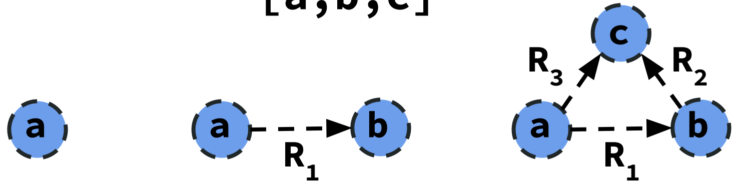
(a, c)
(0, n)
...
(n, 2n)



Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]



(a)
(0)
...
(n)

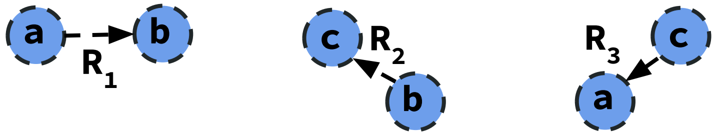
(a, b)
(0, n)
(…, n)

(a, b, c)
(0, n, n+1)

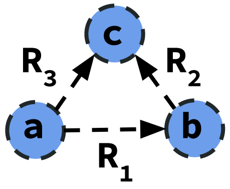
(a, b)
(0, n)
...
(n, 2n)

(b, c)
(0, n)
...
(n, 2n)

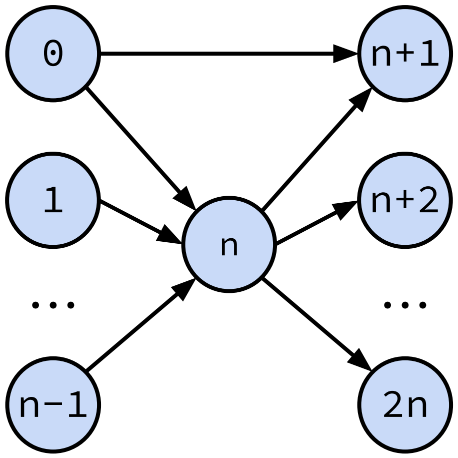
(a, c)
(0, n)
...
(n, 2n)



Query



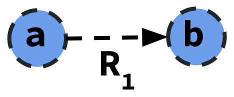
Dataset

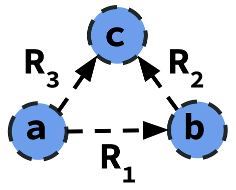


Worst-case Optimal Joins (WCOJs) Example

Join Attribute Ordering (JAO)

[a,b,c]





(a)
(0)
...
(n)

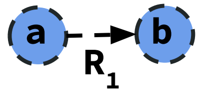
(a, b)
(0, n)
(…, n)

(a, b, c)
(0, n, n+1)

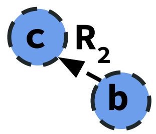
WCOJs correct for sub-optimality
For all join attribute orderings.

→ ***No advice on how to pick JAOs***

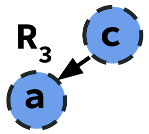
(a, b)
(0, n)
...
(n, 2n)



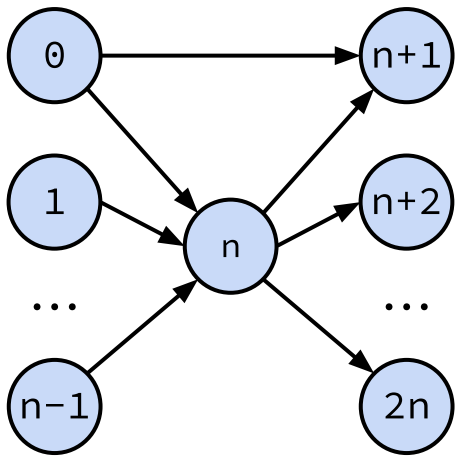
(b, c)
(0, n)
...
(n, 2n)



(a, c)
(0, n)
...
(n, 2n)



<u>Dataset</u>



Research Questions

Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

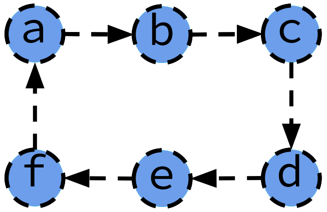
2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



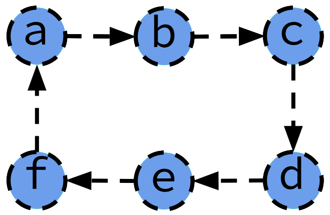
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

WCOJ
Eval.

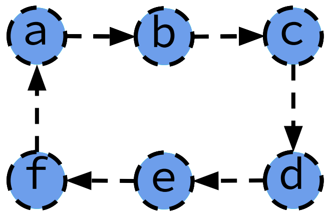


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



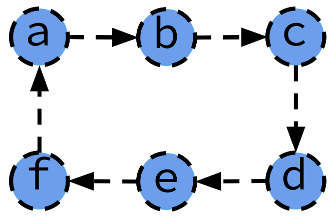
WCOJ
Eval.

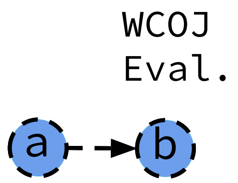
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



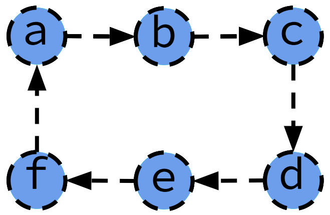


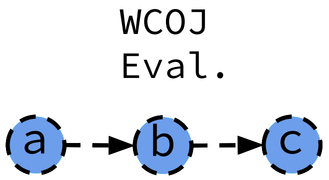
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



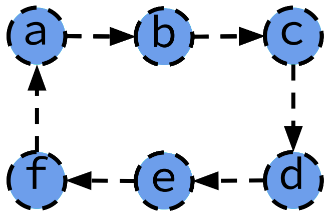


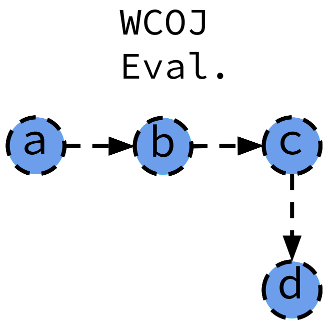
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



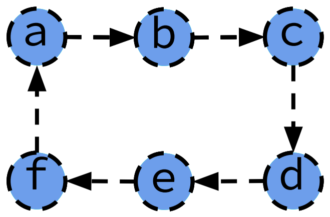


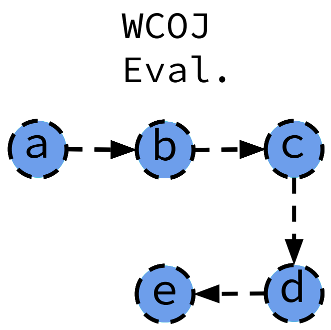
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



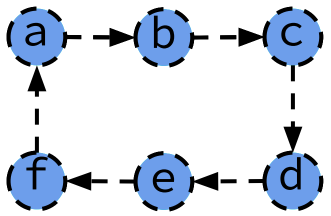


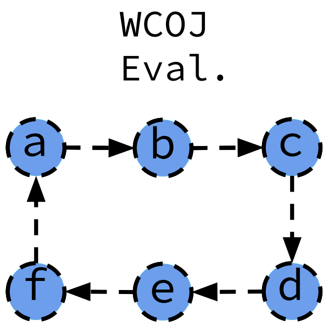
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



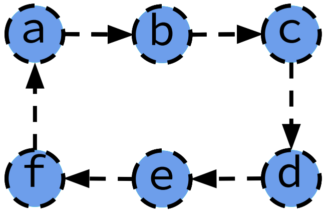


Research Questions

Given an input Query,

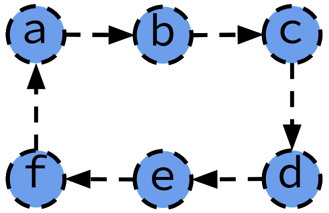
1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



WCOJ
Eval.

Binary Join
Eval.



Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

$$\begin{tikzcd}
a \arrow[r, dashed] & b \arrow[r, dashed] & c \arrow[d, dashed] \\
f \arrow[u, dashed] & e \arrow[l, dashed] & d \arrow[l, dashed]
\end{tikzcd}$$

WCOJ
Eval.

$$\begin{tikzcd}
a \arrow[r, dashed] & b \arrow[r, dashed] & c \arrow[d, dashed] \\
f \arrow[u, dashed] & e \arrow[l, dashed] & d \arrow[l, dashed]
\end{tikzcd}$$

Binary Join
Eval.

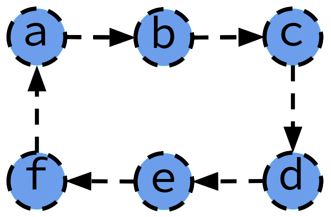
$$\begin{tikzcd}
a \arrow[r, dashed] & b
\end{tikzcd}$$

Research Questions

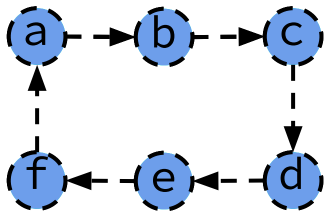
Given an input Query,

1) How to pick good join attribute orderings?

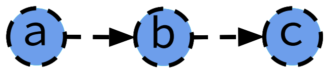
2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



WCOJ Eval.



Binary Join Eval.

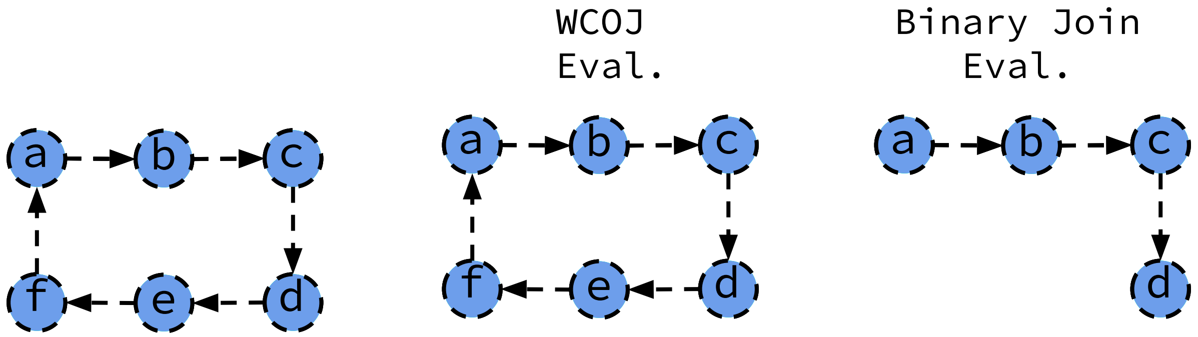


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

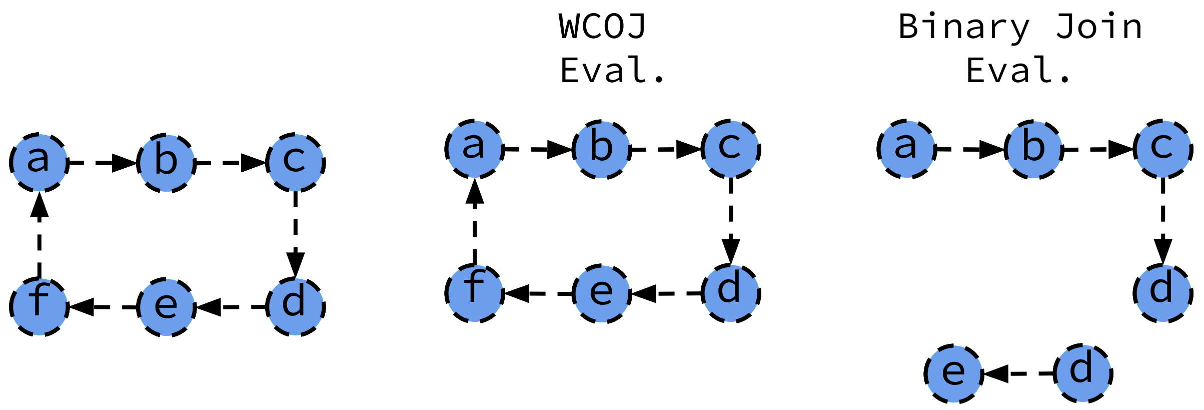


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

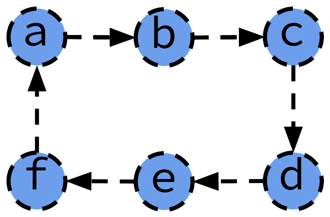


Research Questions

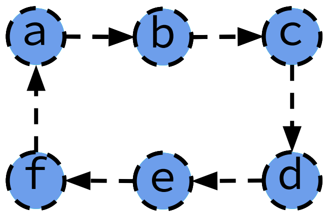
Given an input Query,

1) How to pick good join attribute orderings?

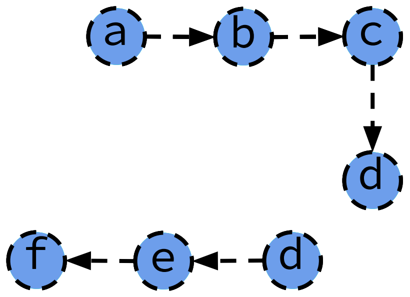
2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



WCOJ
Eval.



Binary Join
Eval.

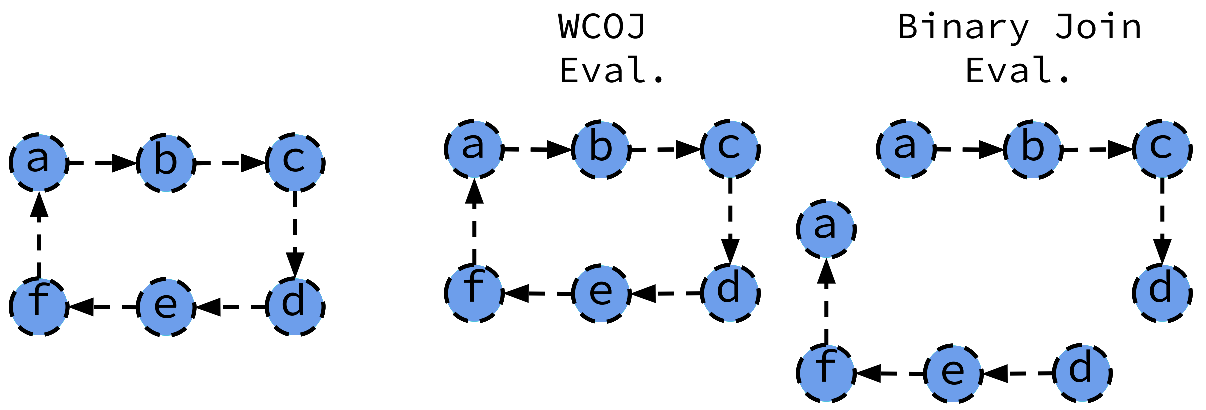


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

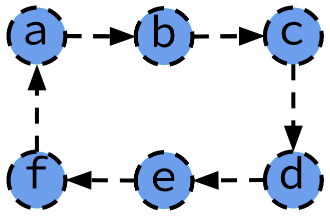


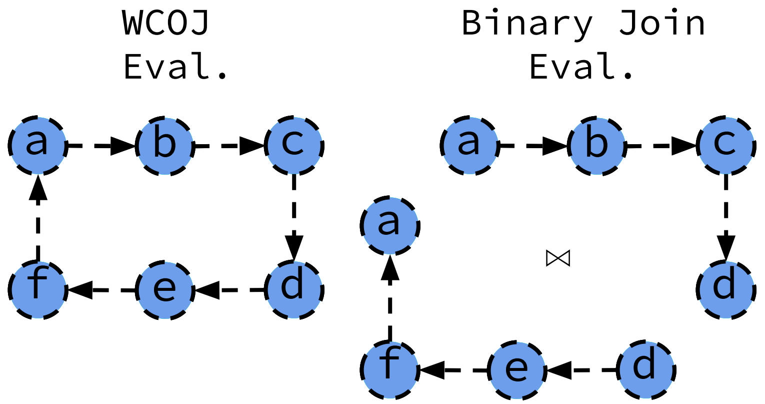
Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



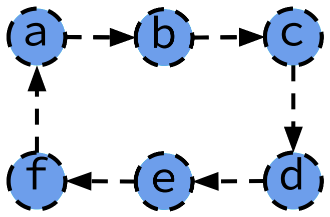


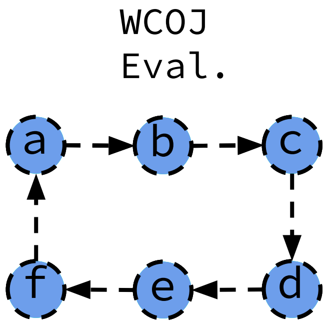
Research Questions

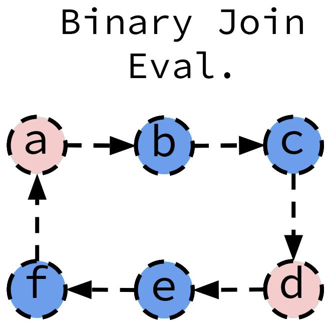
Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?





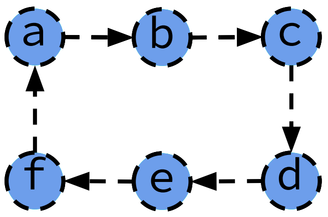


Research Questions

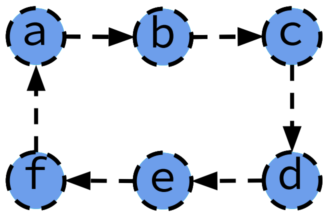
Given an input Query,

1) How to pick good join attribute orderings?

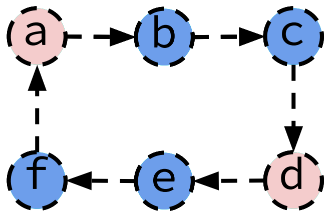
2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



WCOJ Eval.



Binary Join Eval.



Hybrid Eval.

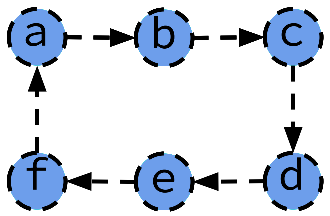
Research Questions

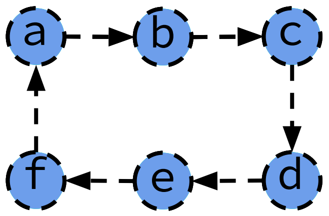
Given an input Query,

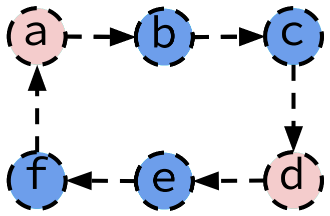
1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

WCOJ Eval. | Binary Join Eval. | Hybrid Eval.





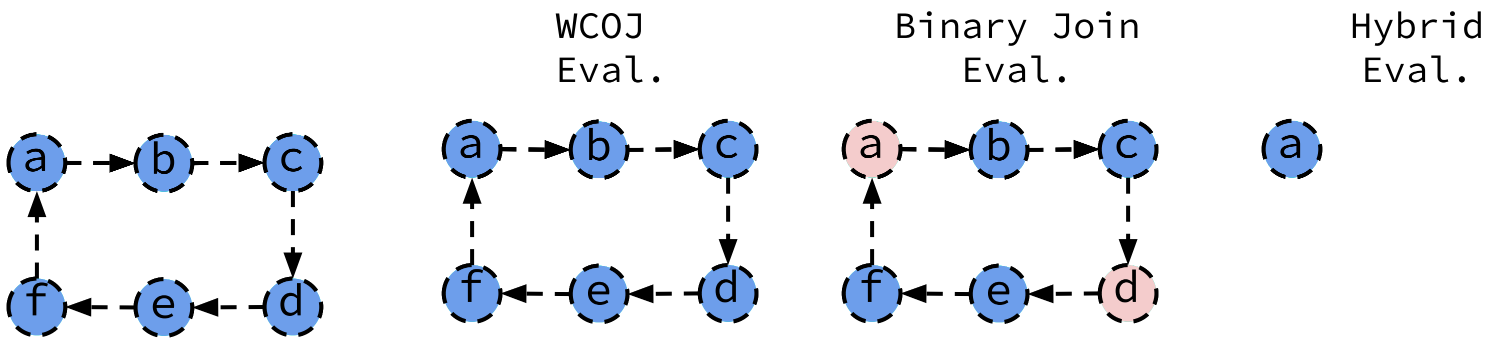


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

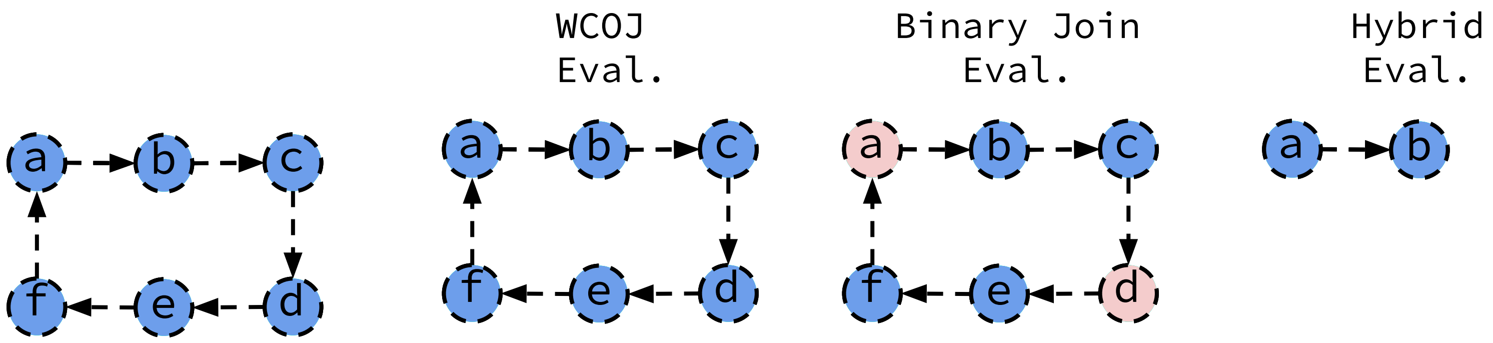


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

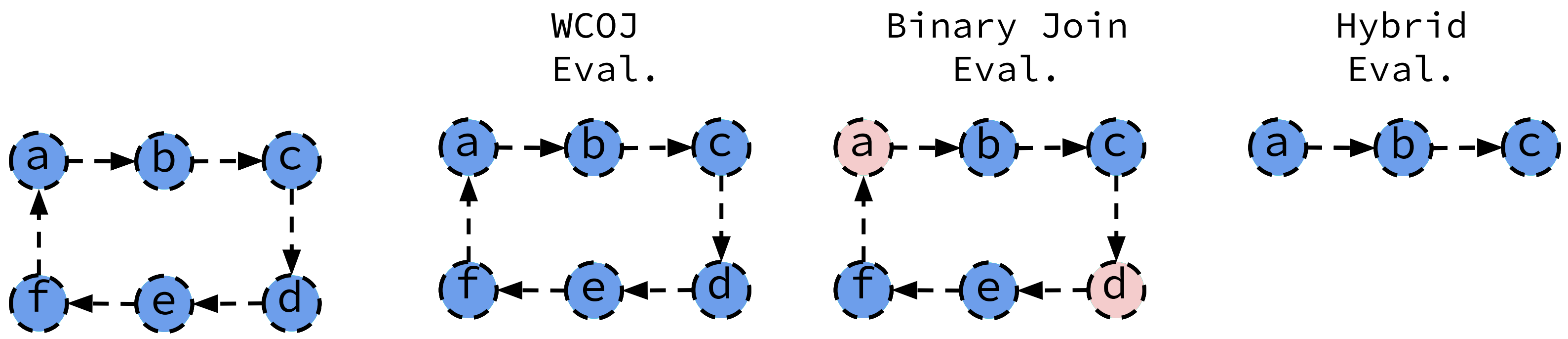


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

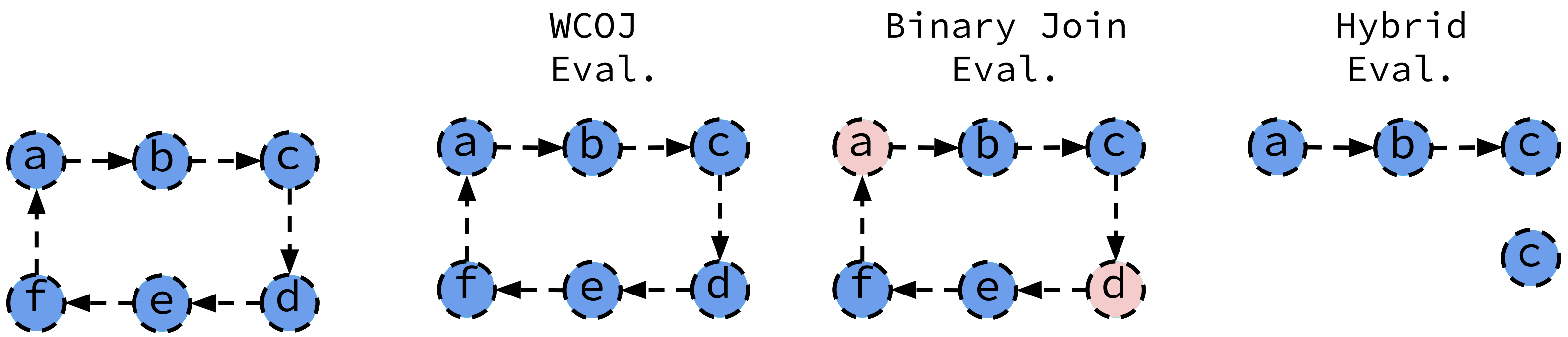


Research Questions

Given an input Query,

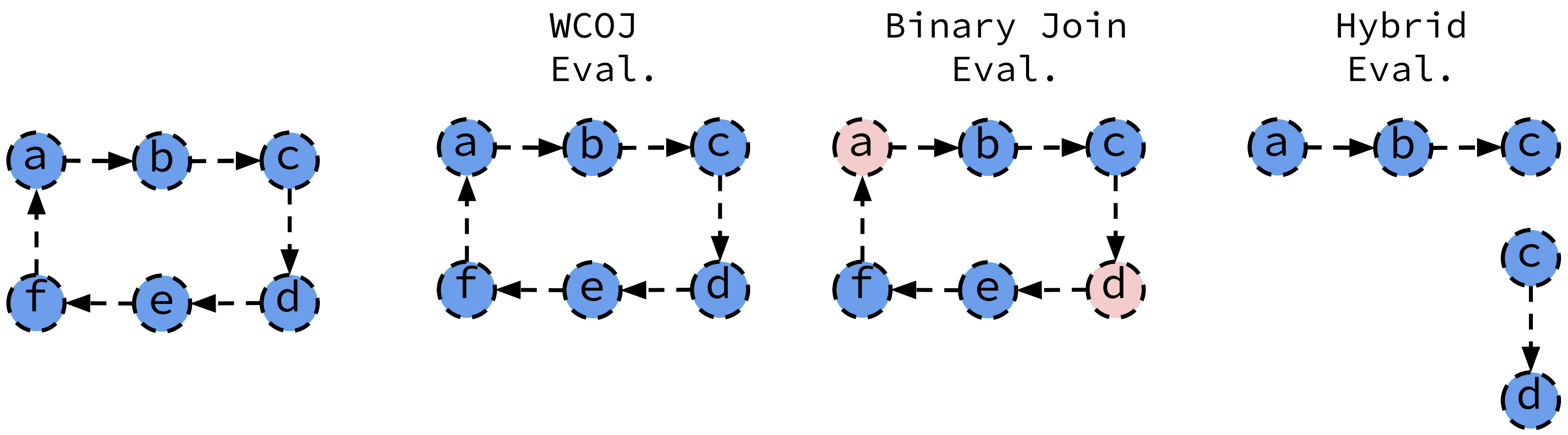
1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



Research Questions

```
Given an input Query,

1)  How to pick good join attribute orderings?

2)  How to generate efficient plans that mix binary
    joins and worst-case optimal joins (WCOJs)?
```

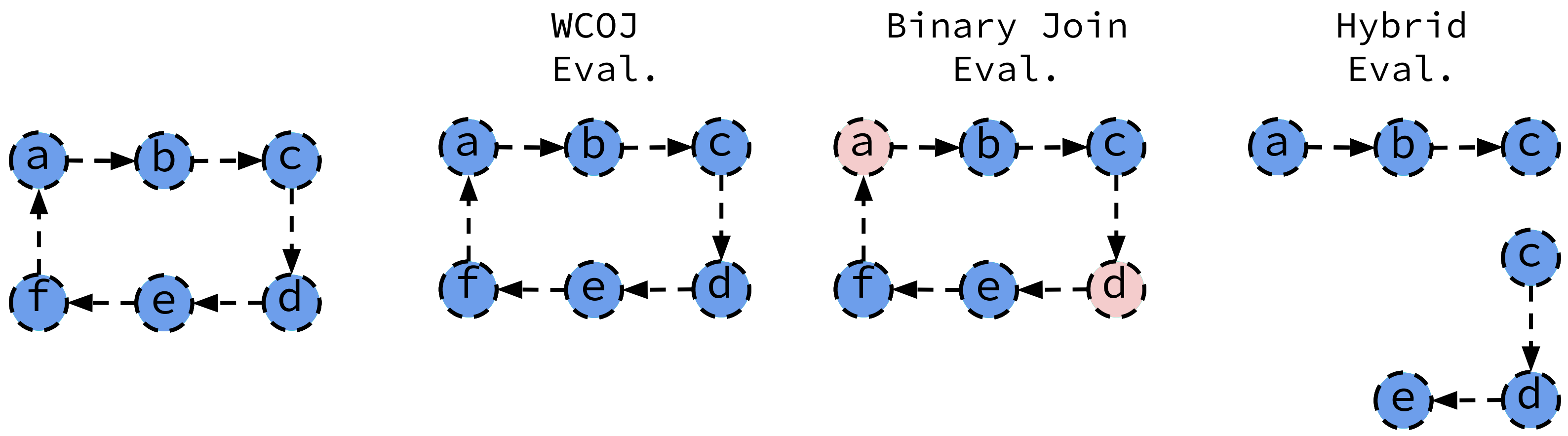


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

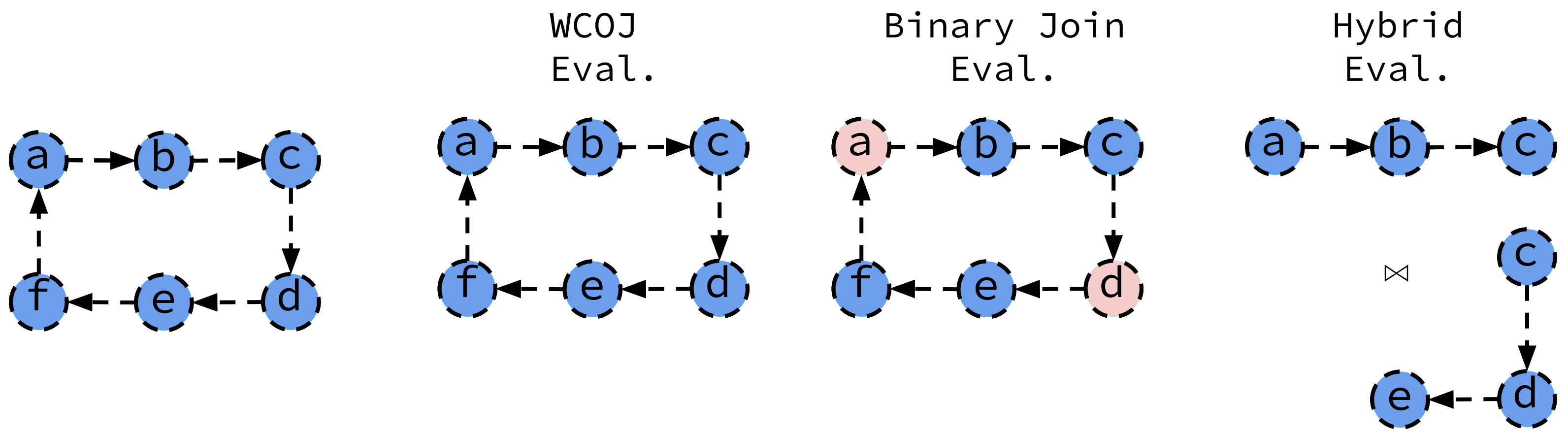


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

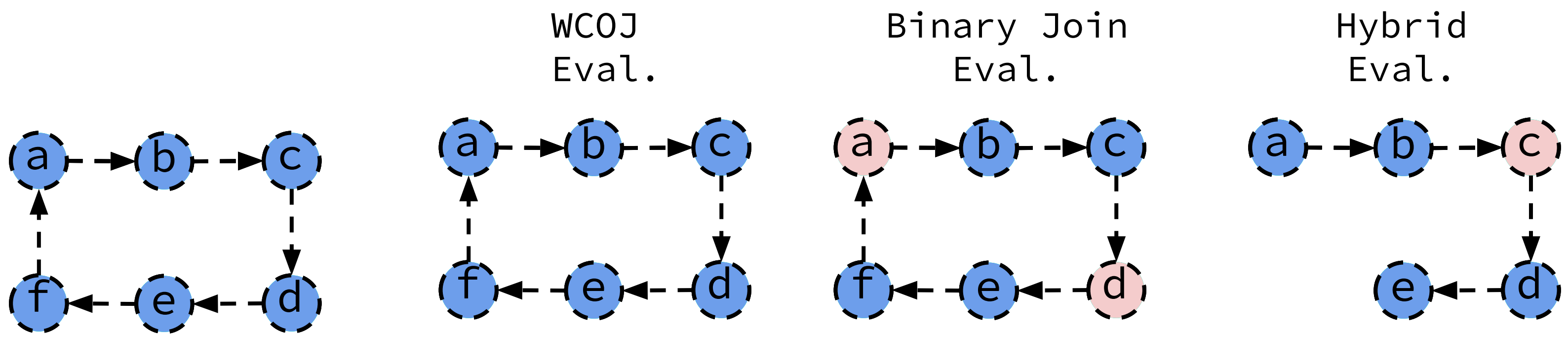


Research Questions

Given an input Query,

1) How to pick good join attribute orderings?

2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?

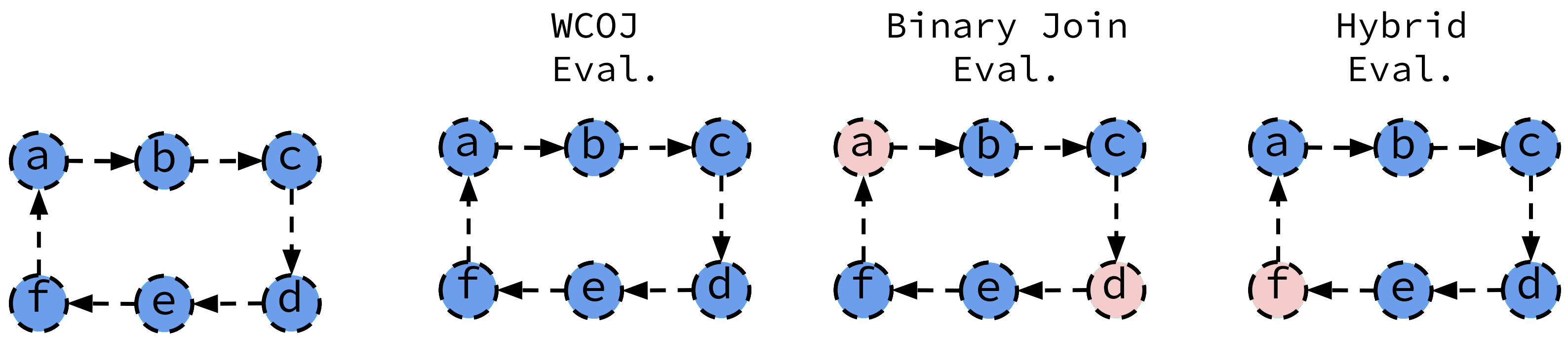


Research Questions

Given an input Query,

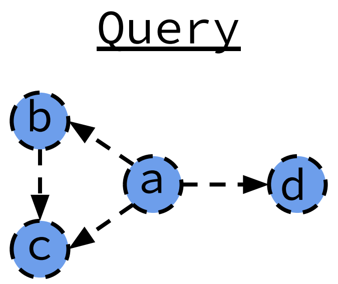
1) How to pick good join attribute orderings?

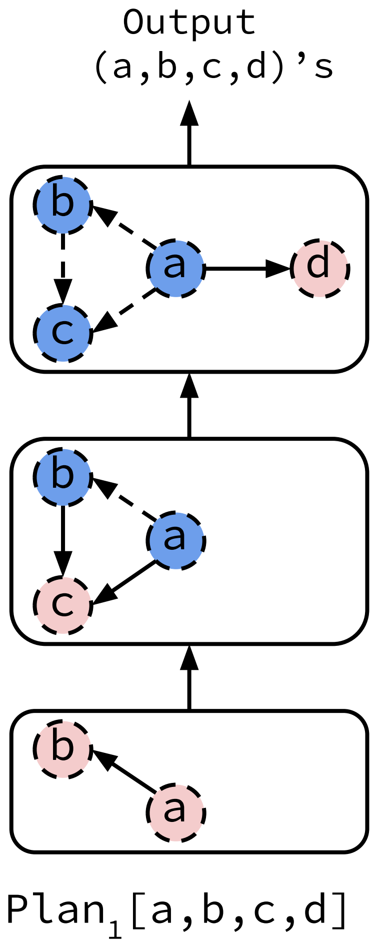
2) How to generate efficient plans that mix binary joins and worst-case optimal joins (WCOJs)?



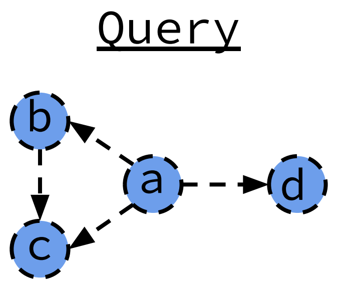
How to Pick Good Join Attribute Ordering (JAO)?

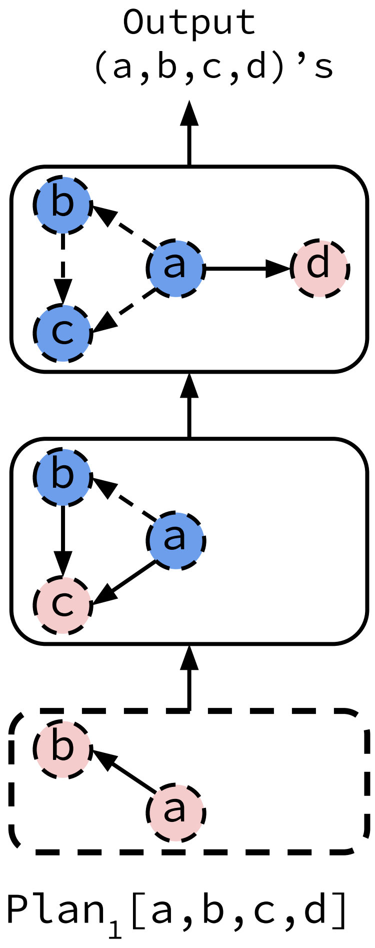
WCOJ Evaluation Overview





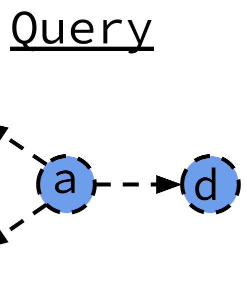
WCOJ Evaluation Overview

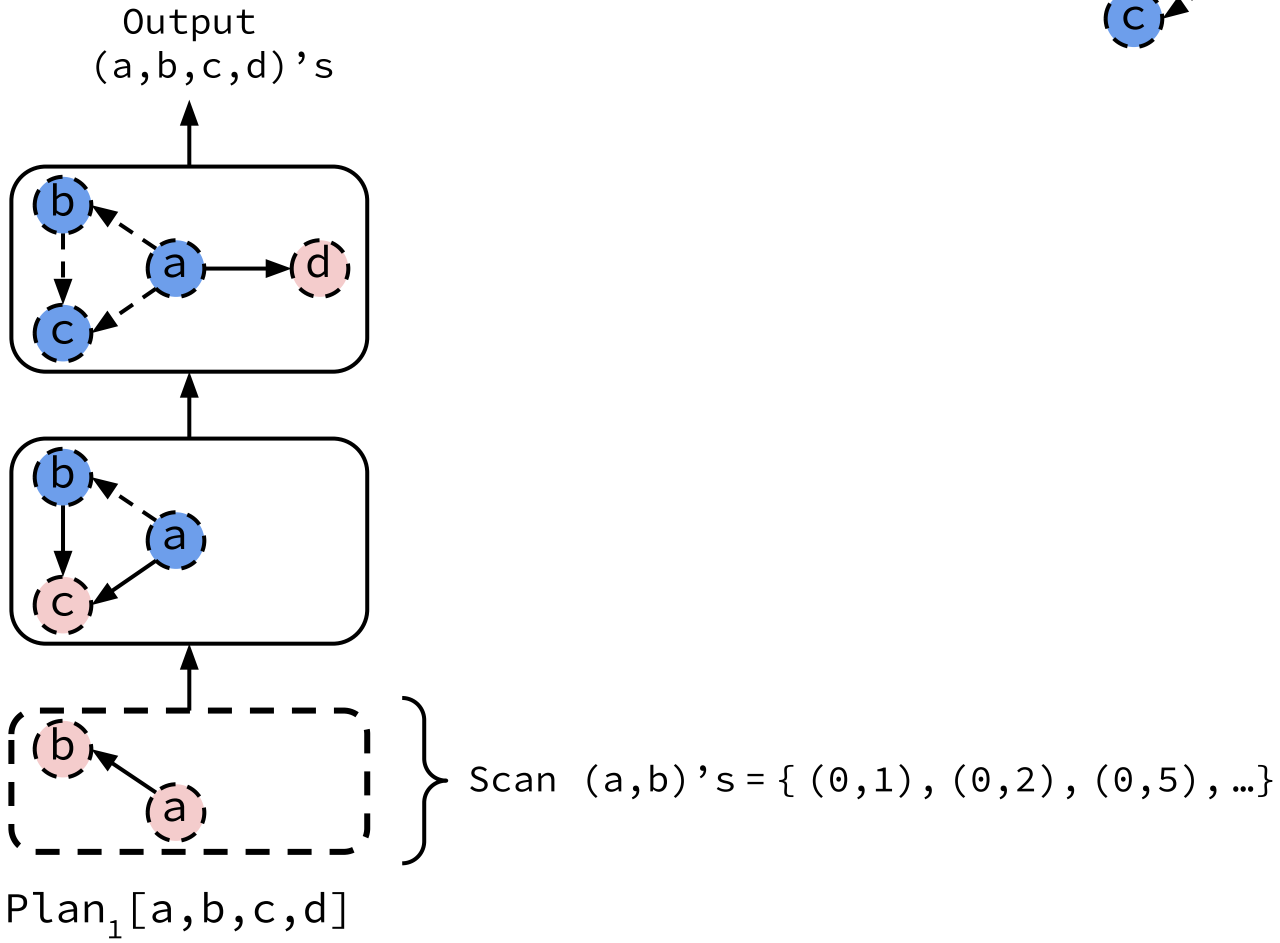




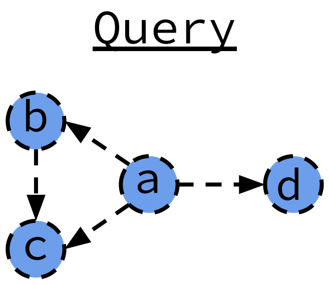
Plan$_1$[a,b,c,d]

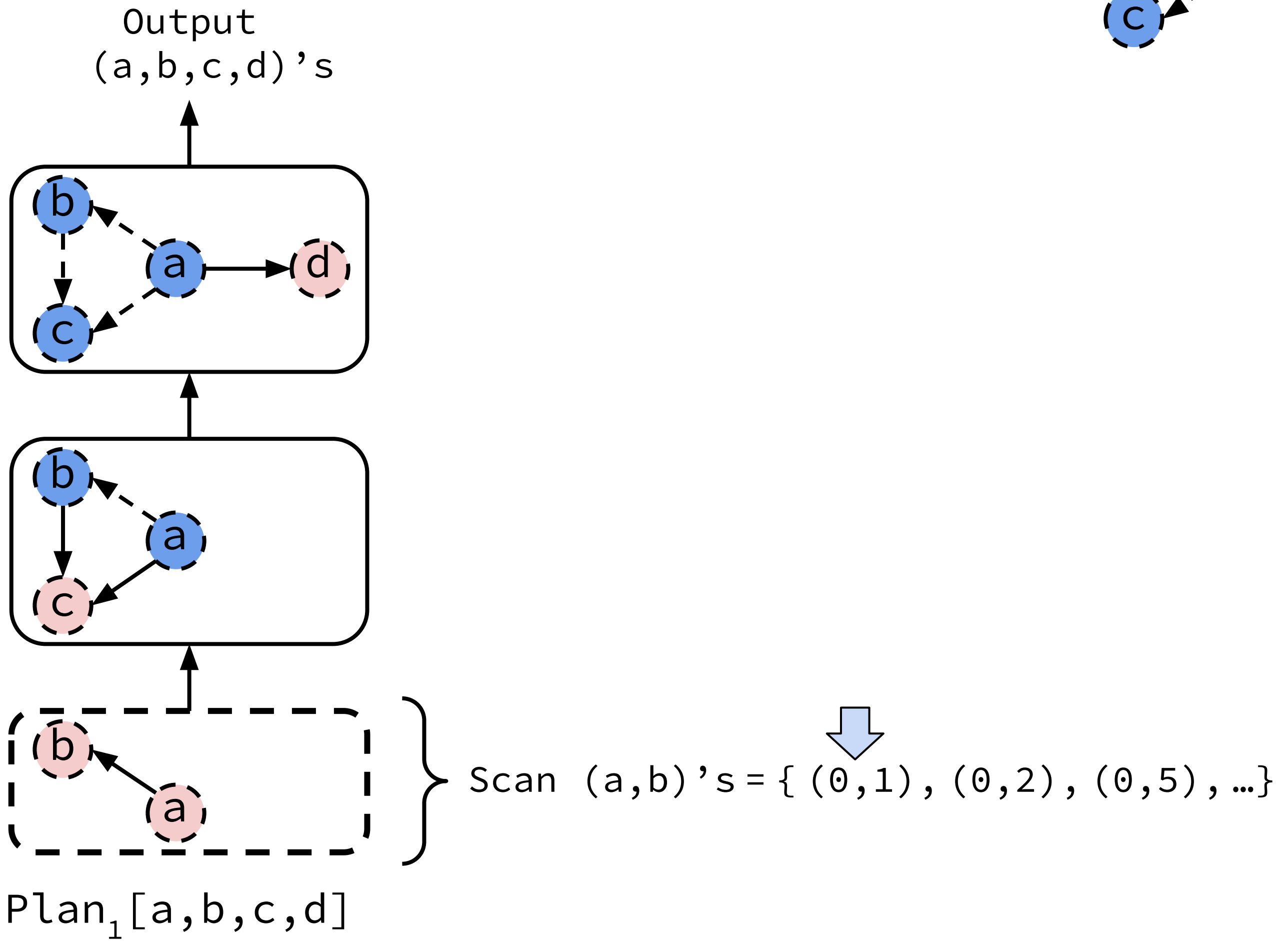
WCOJ Evaluation Overview





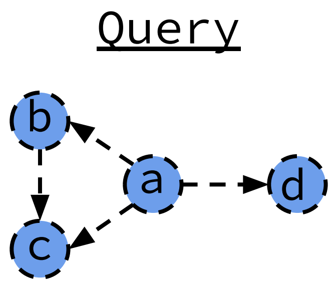
WCOJ Evaluation Overview

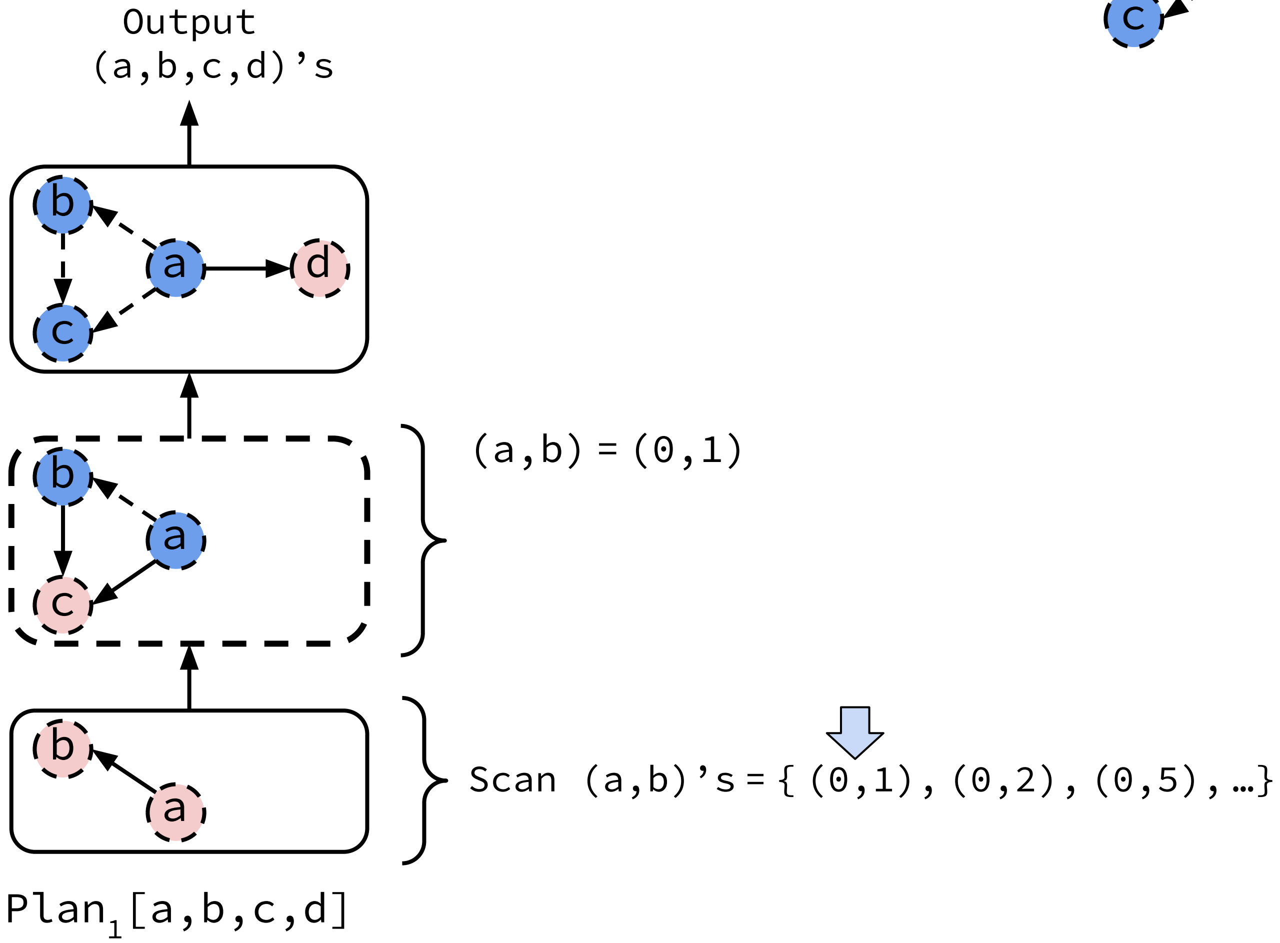




Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

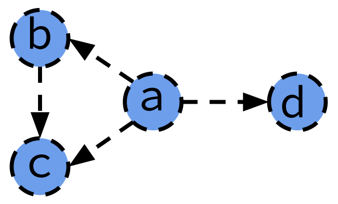
WCOJ Evaluation Overview



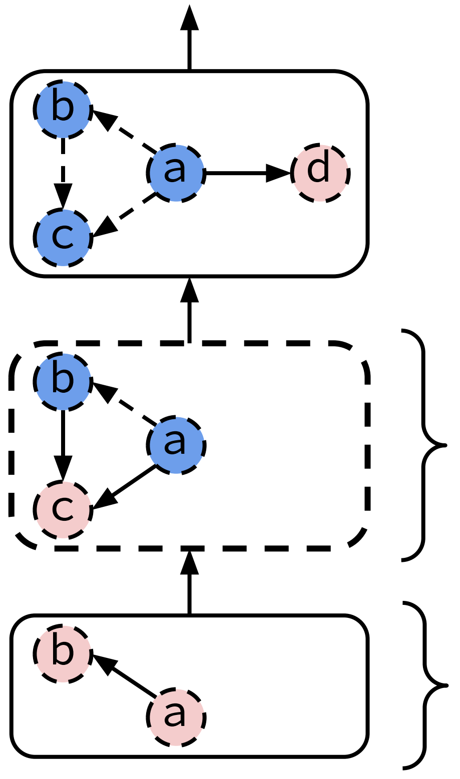


WCOJ Evaluation Overview

Query



Output
(a,b,c,d)'s

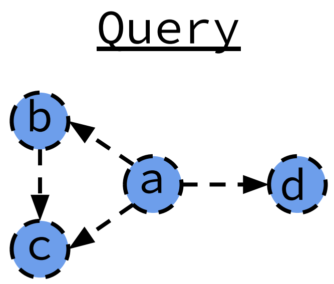


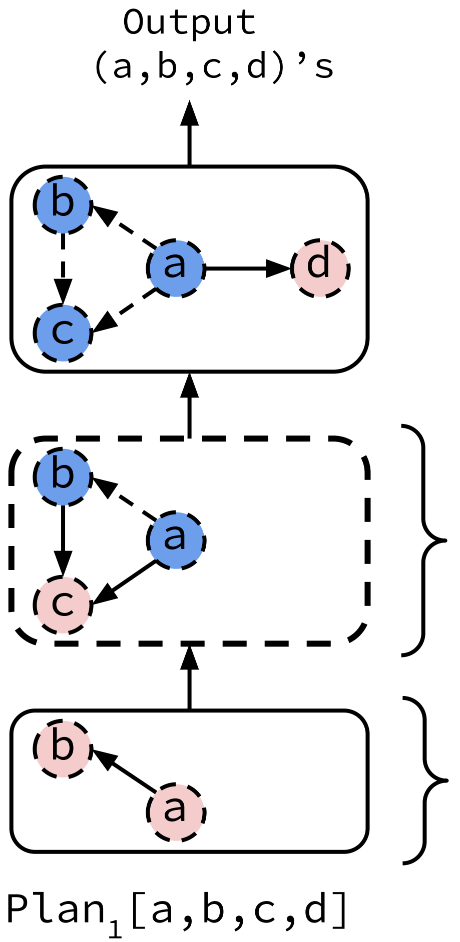
(a,b) = (0,1)
c's: fwd(a = 0) ∩ fwd(b = 1)
{1, 2, 5, …} ∩ {2, 3, 6, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

Plan$_1$[a,b,c,d]

WCOJ Evaluation Overview





(a,b) = (0,1)

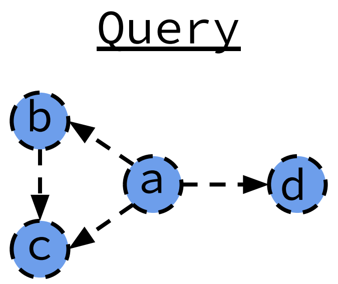
c's: fwd(a = 0) ∩ fwd(b = 1)

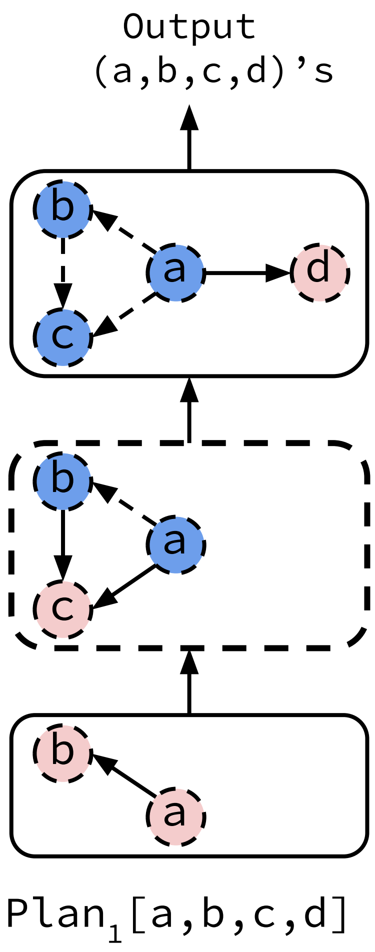
{1, 2, 5, …} ∩ {2, 3, 6, …}

{ 2, 7, 9, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview





(a,b) = (0,1)

c's: fwd(a = 0) ∩ fwd(b = 1)

{1, 2, 5, …} ∩ {2, 3, 6, …}

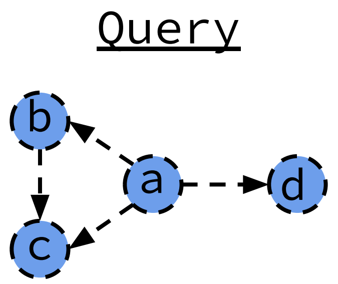
⇩

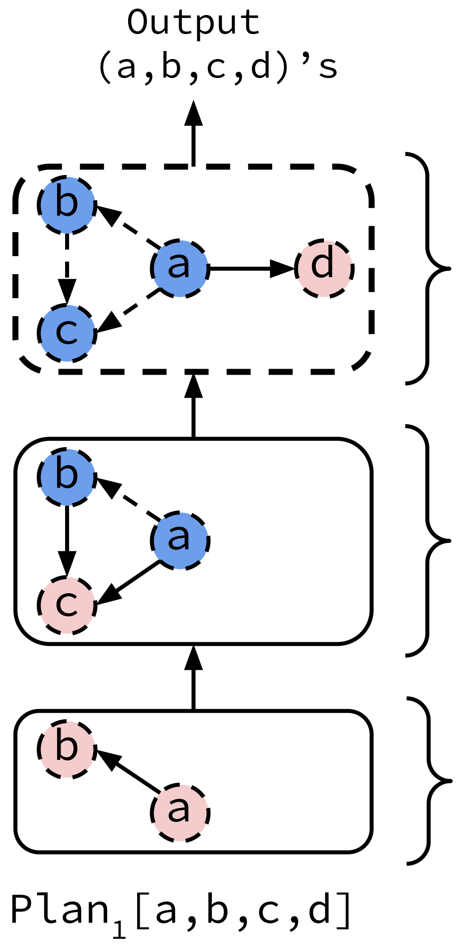
{ 2, 7, 9, …}

⇩

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview





Plan$_1$[a,b,c,d]

(a,b,c) = (0,1,2)

(a,b) = (0,1)

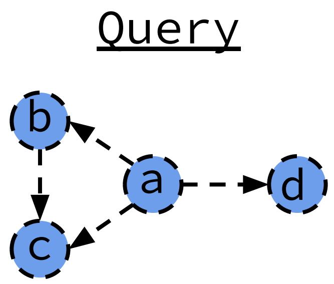
c's: fwd(a = 0) ∩ fwd(b = 1)

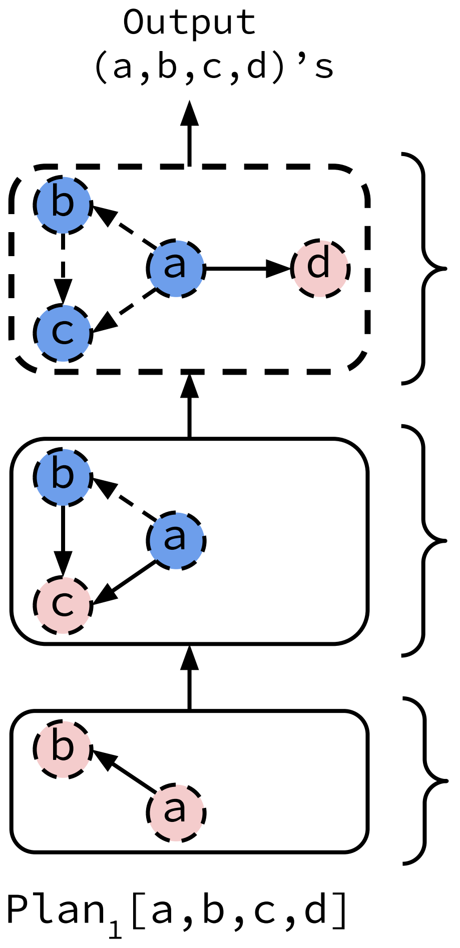
{1, 2, 5, …} ∩ {2, 3, 6, …}

{ 2, 7, 9, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview





(a,b,c) = (0,1,2)

d's: fwd(a=0): {1, 2, 5, …}

(a,b) = (0,1)

c's: fwd(a=0) ∩ fwd(b=1)

{1, 2, 5, …} ∩ {2, 3, 6, …}

⇩

{2, 7, 9, …}

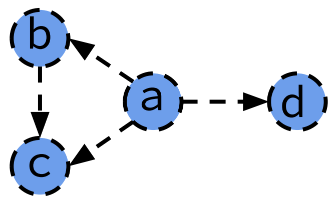
⇩

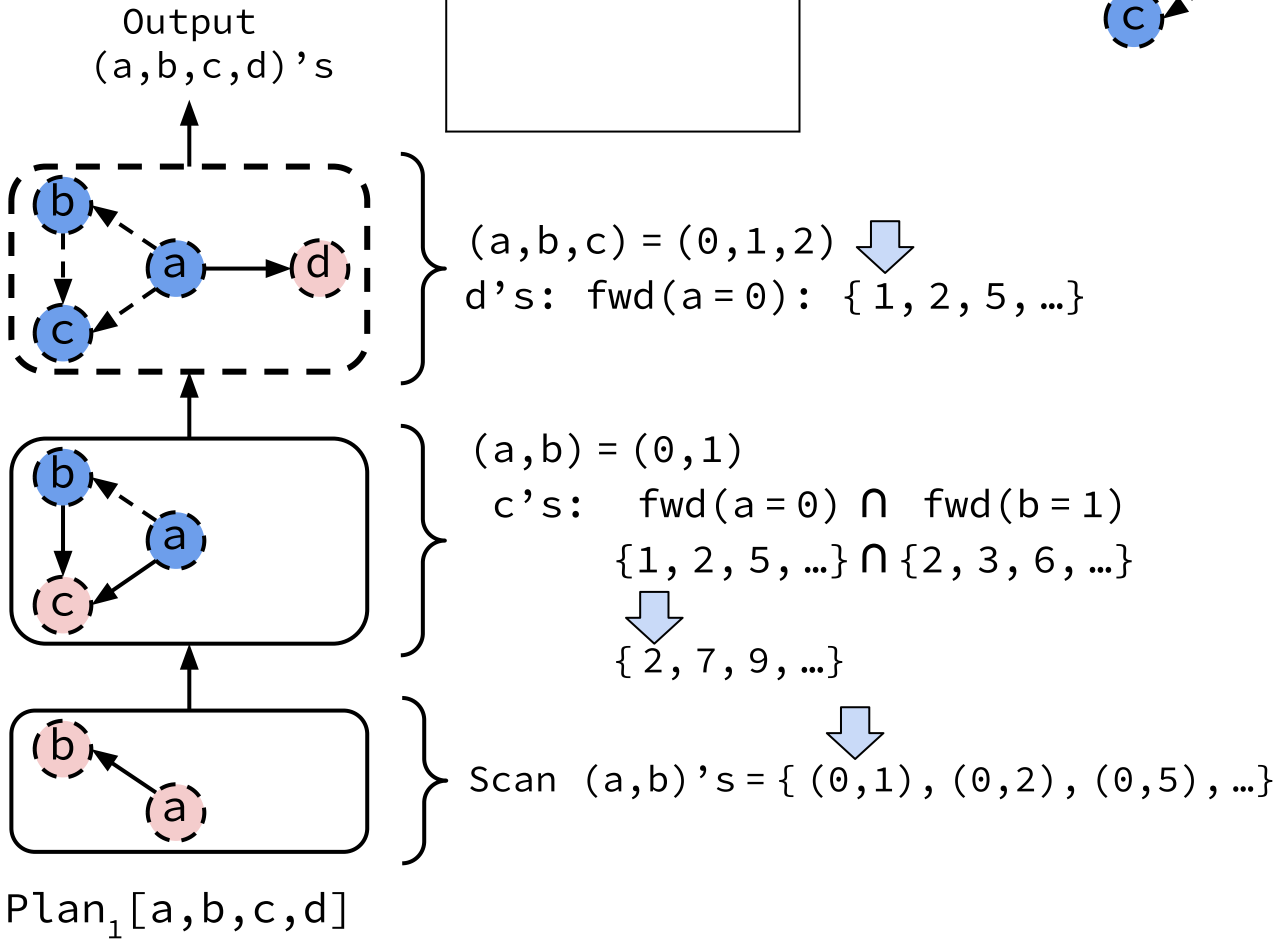
Scan (a,b)'s = {(0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview

(a,b,c,d)
(0, 1, 2, 1)

Query

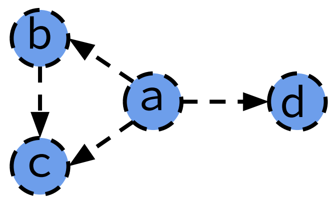


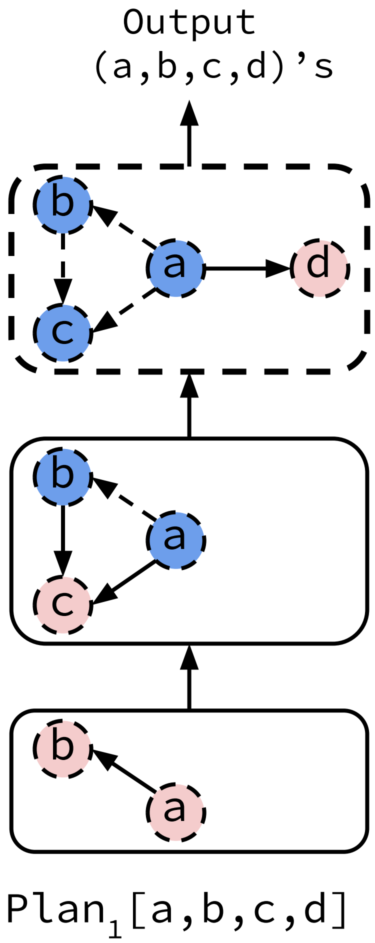


WCOJ Evaluation Overview

(a,b,c,d)
(0, 1, 2, 1)
(0, 1, 2, 2)

Query





Plan$_1$[a,b,c,d]

(a,b,c) = (0,1,2)

d's: fwd(a = 0): { 1, 2, 5, …}

(a,b) = (0,1)

c's: fwd(a = 0) ∩ fwd(b = 1)

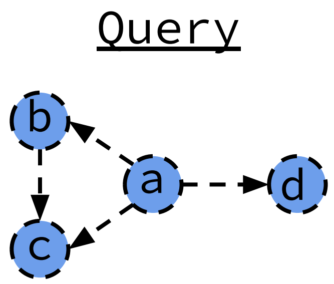
{1, 2, 5, …} ∩ {2, 3, 6, …}

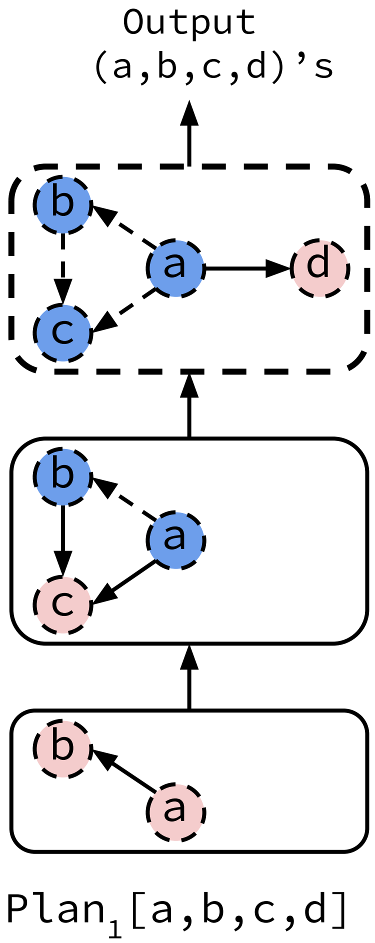
{ 2, 7, 9, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview

(a,b,c,d)
(0, 1, 2, 1)
(0, 1, 2, 2)
(0, 1, 2, 5)





(a,b,c) = (0,1,2)

d's: fwd(a = 0): { 1, 2, 5, …}

(a,b) = (0,1)

c's: fwd(a = 0) ∩ fwd(b = 1)

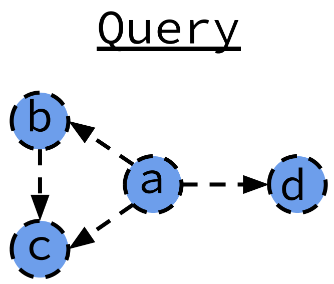
{1, 2, 5, …} ∩ {2, 3, 6, …}

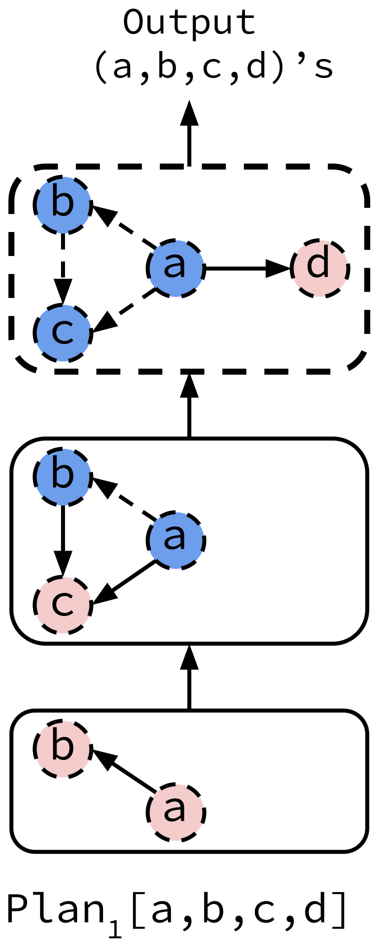
{ 2, 7, 9, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview

(a,b,c,d)
(0, 1, 2, 1)
(0, 1, 2, 2)
(0, 1, 2, 5)
...





(a,b,c) = (0,1,2)

d's: fwd(a = 0): { 1, 2, 5, …}

(a,b) = (0,1)

c's: fwd(a = 0) ∩ fwd(b = 1)

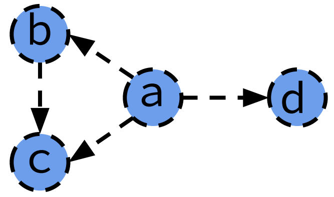
{1, 2, 5, …} ∩ {2, 3, 6, …}

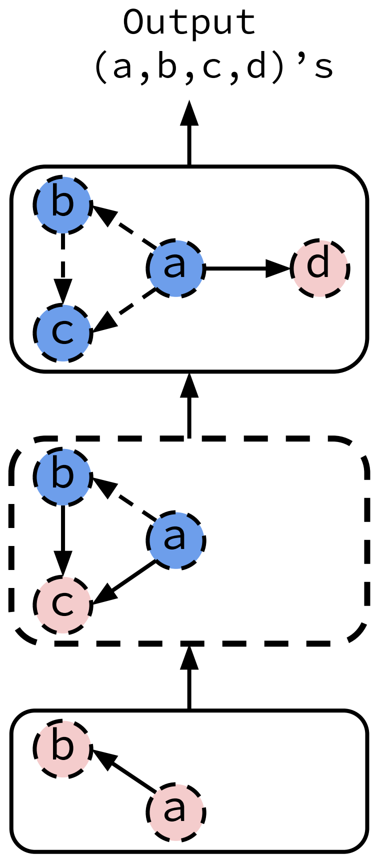
{ 2, 7, 9, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

WCOJ Evaluation Overview

(a,b,c,d)
(0, 1, 2, 1)
(0, 1, 2, 2)
(0, 1, 2, 5)
...





(a,b) = (0,1)

c's: fwd(a = 0) ∩ fwd(b = 1)

{1, 2, 5, …} ∩ {2, 3, 6, …}

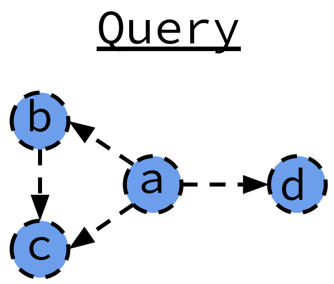
{ 2, 7, 9, …}

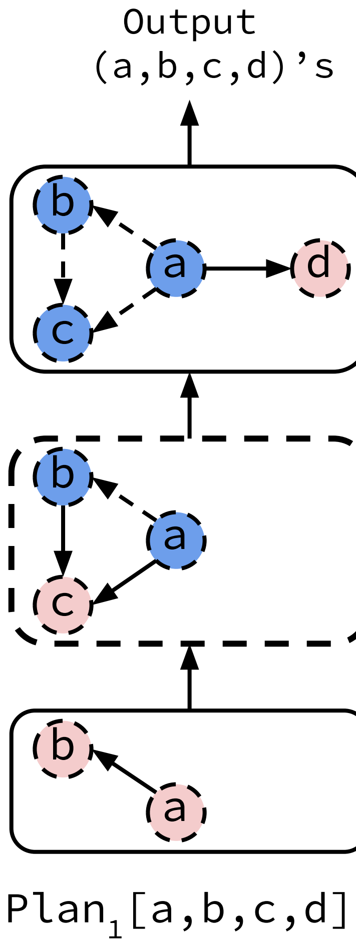
Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

$\text{Plan}_1[a,b,c,d]$

WCOJ Evaluation Overview

(a,b,c,d)
(0, 1, 2, 1)
(0, 1, 2, 2)
(0, 1, 2, 5)
...





(a,b) = (0,1)

c's: fwd(a = 0) ∩ fwd(b = 1)

{1, 2, 5, …} ∩ {2, 3, 6, …}

{ 2, 7, 9, …}

Scan (a,b)'s = { (0,1), (0,2), (0,5), …}

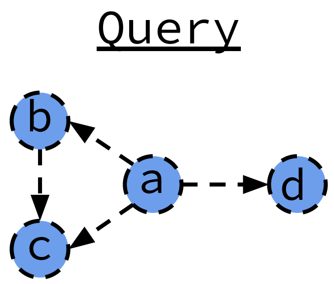
How to Pick Good Join Attribute Ordering (JAO)?

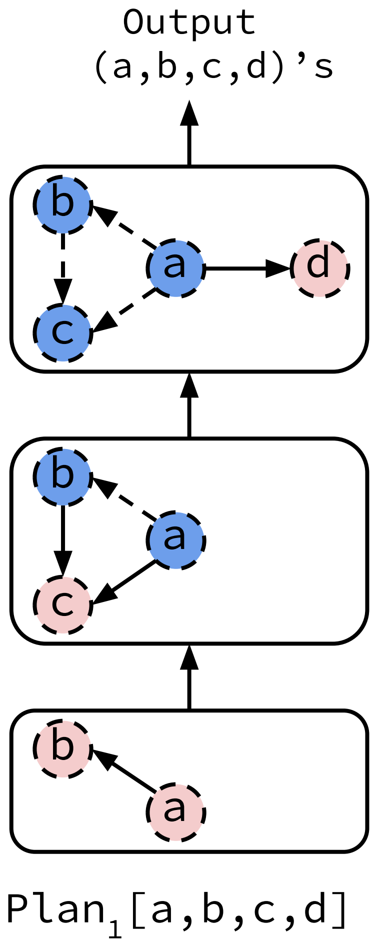
```
Impact of JAO:
1) Number of intermediate results
2) Direction of Adj. Lists Intersected
```

How to Pick Good Join Attribute Ordering (JAO)?

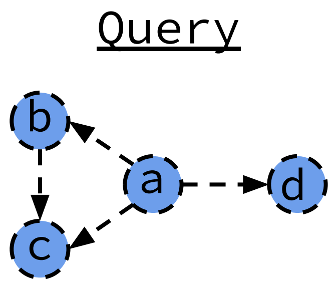
```
Impact of JAO:
1) Number of intermediate results
2) Direction of Adj. Lists Intersected
```

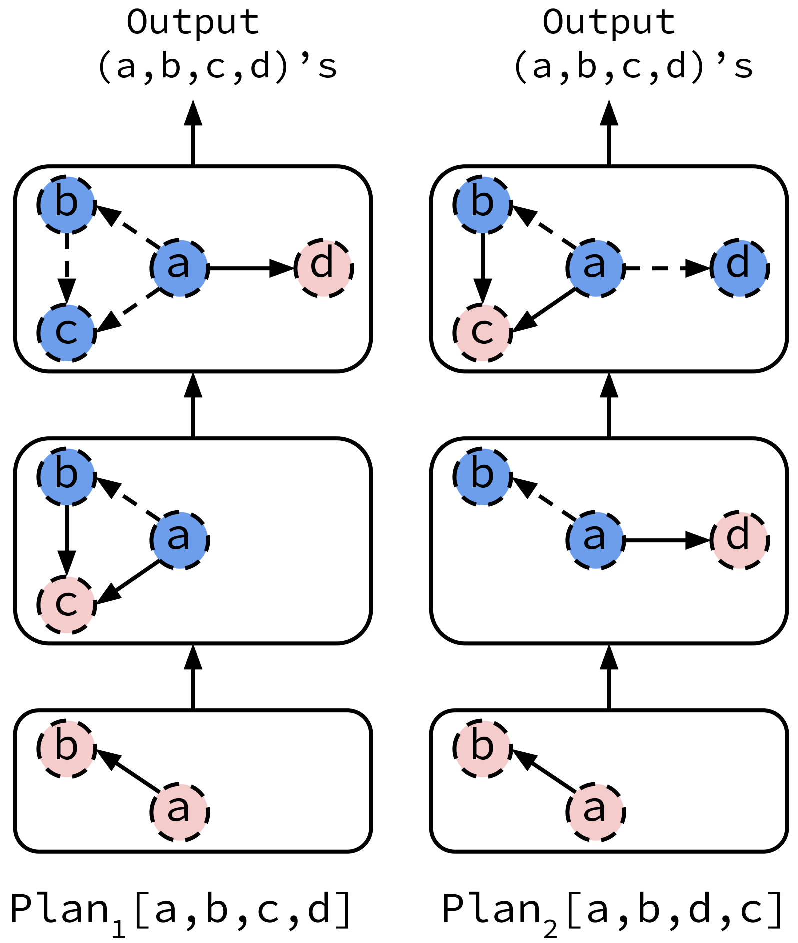
1. JAO → Number of intermediate results





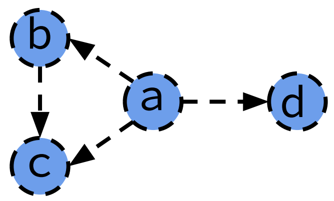
1. JAO → Number of intermediate results

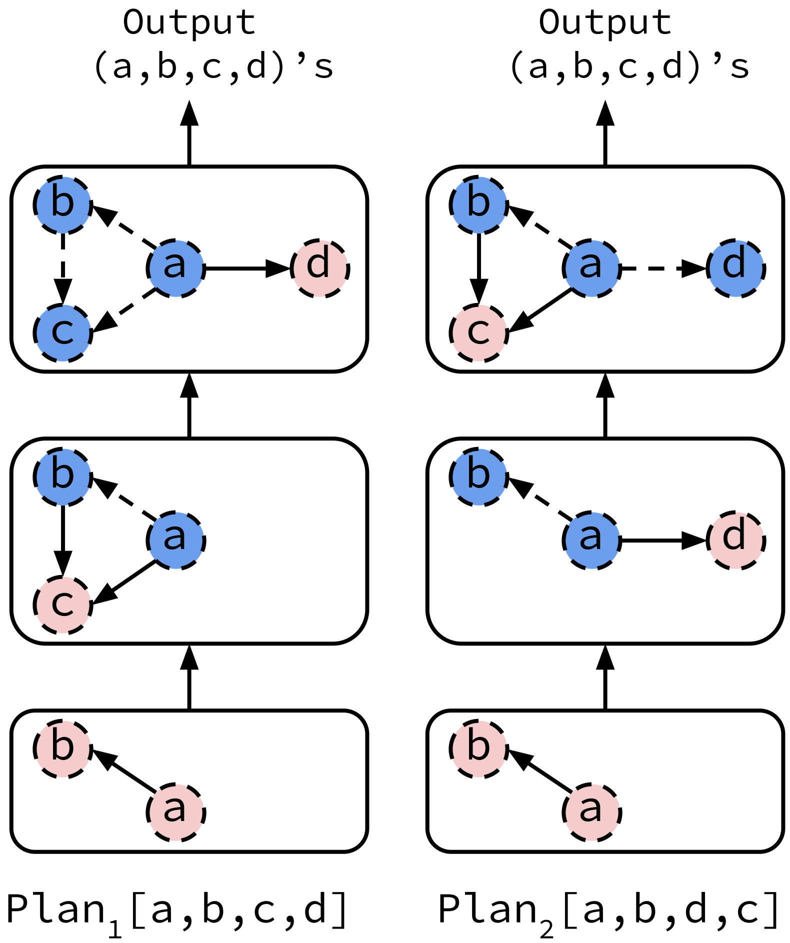




1. JAO → Number of intermediate results

Query





Dataset	Plan	# Int. Results	Runtime (secs)
Epinions V=75K E=508K	Plan_1	4M	0.9
	Plan_2	55M **(13.8x)**	56.6 **(62.9x)**

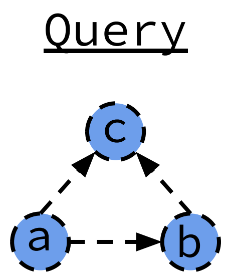
How to Pick Good Join Attribute Ordering (JAO)?

Impact of JAO:

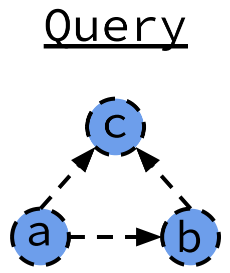
1) Number of intermediate results

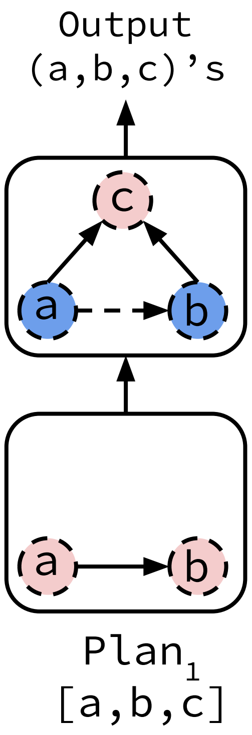
2) Direction of Adj. Lists Intersected

2. JAO → Direction of Adj. Lists Intersected

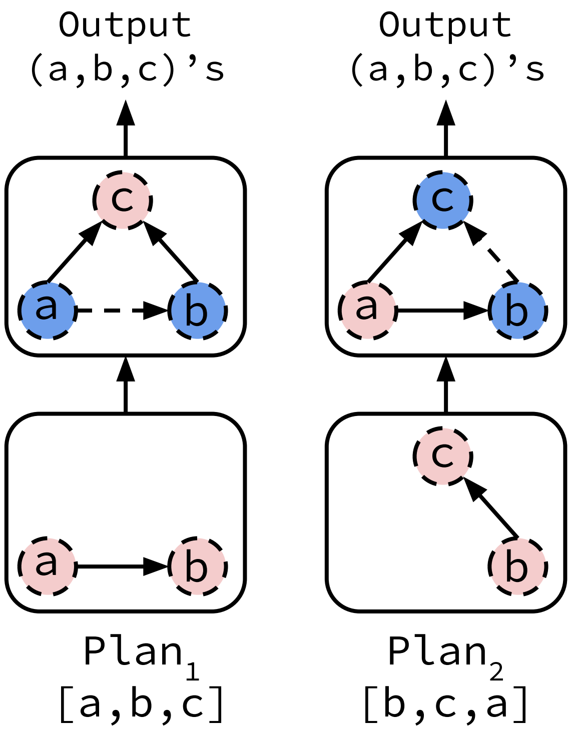


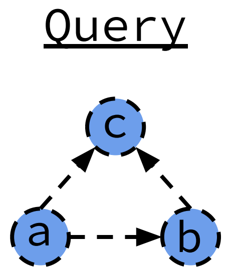
2. JAO → Direction of Adj. Lists Intersected



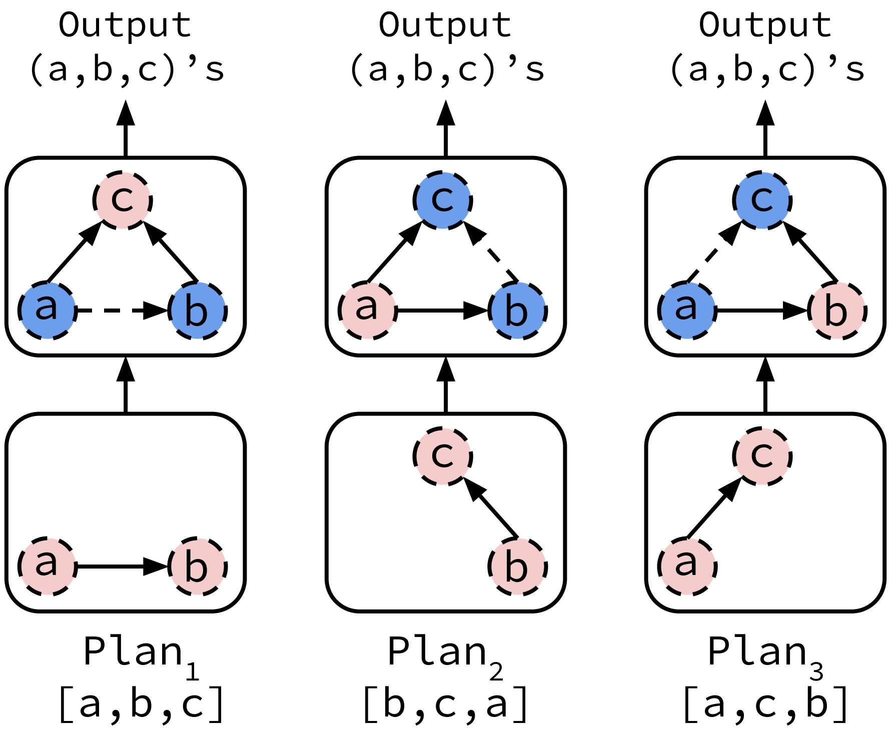


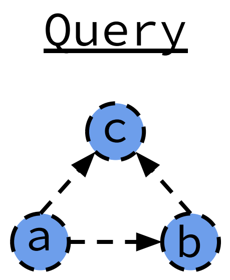
2. JAO → Direction of Adj. Lists Intersected



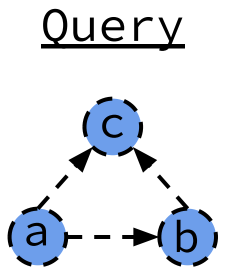


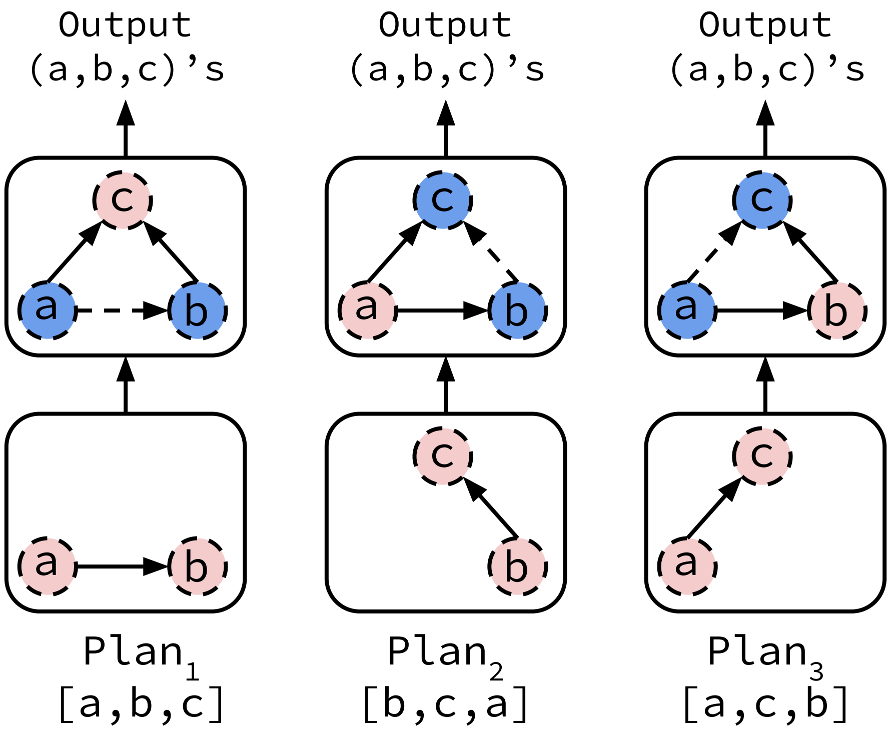
2. JAO → Direction of Adj. Lists Intersected





2. JAO → Direction of Adj. Lists Intersected





Dataset	Plan	Σ Adj List sizes	Runtime (secs)
BerkStan V=685K E=7.6M	Plan$_1$	0.5B	2.6
	Plan$_2$	55B **(113.8x)**	15.2 **(5.8x)**
	Plan$_3$	55B **(114.0x)**	31.6 **(12.2x)**

How to Pick Good Join Attribute Ordering (JAO)?

Impact of JAO:

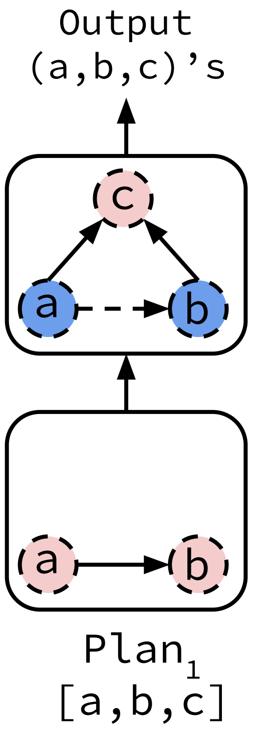
1) Number of intermediate results

2) Direction of Adj. Lists Intersected

→ Account for impact in ***cost model***

Cost Metric - Intersection Cost (I-Cost)

Cost of a WCOJ plan is total **intersection-cost** of all operators.
I-cost: size of intersected adj lists throughout execution.



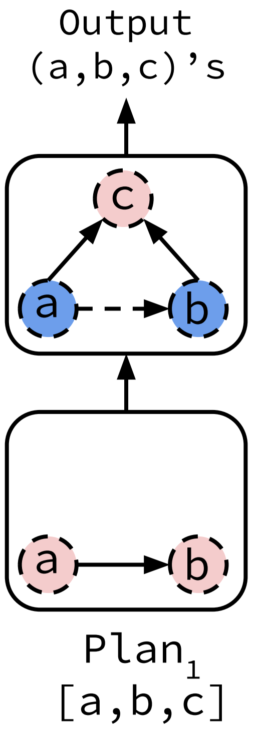
Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

Within an operator, consider each tuple

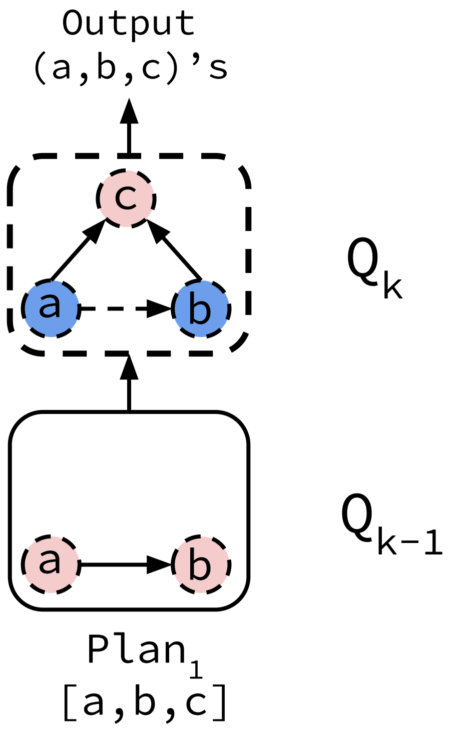
$$\sum_{t \in Q_{k-1}}$$



Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.
I-cost: size of intersected adj lists throughout execution.

Within an operator, consider each tuple



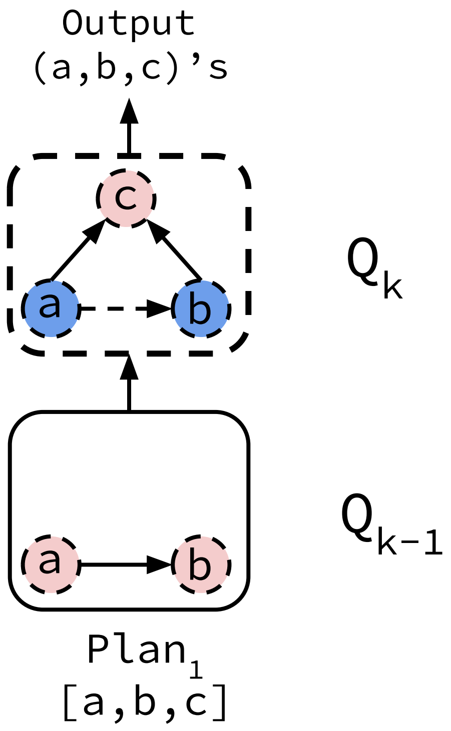
$$\sum_{t \in Q_{k-1}}$$

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

Within an operator, consider each tuple



$$\sum_{t \in Q_{k-1}}$$

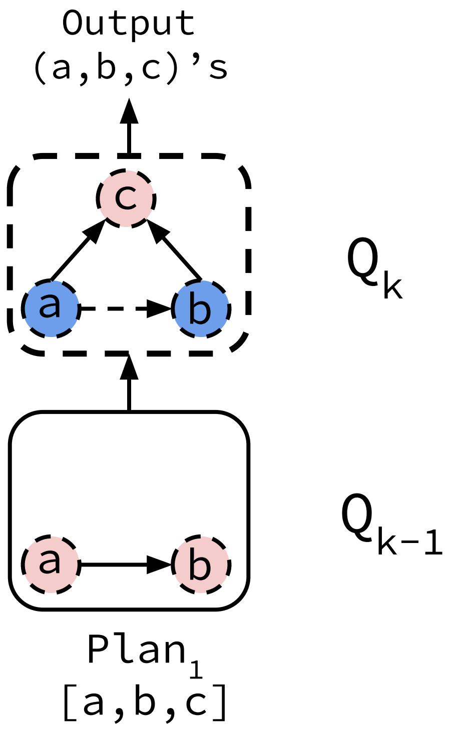
t = (a,b)
(0, 1)
(0, 2)
(1, 5)
...

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

Within an operator, consider each tuple's adj lists



$$\sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|$$

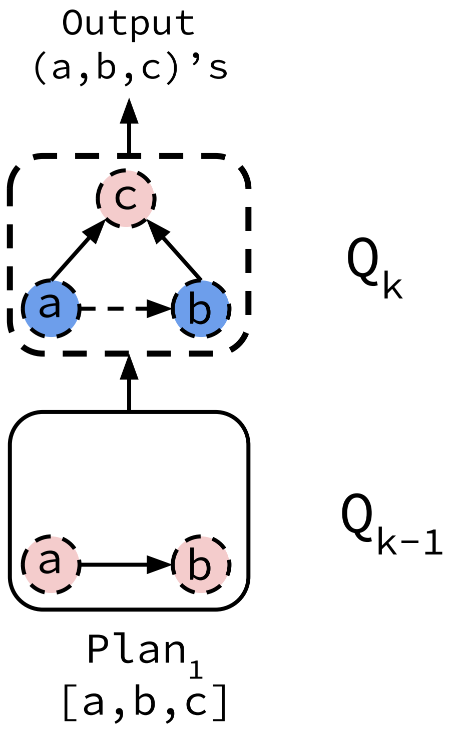
t = (a,b)
(0, 1)
(0, 2)
(1, 5)
...

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

Within an operator, consider the # input tuples



$$\sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|$$

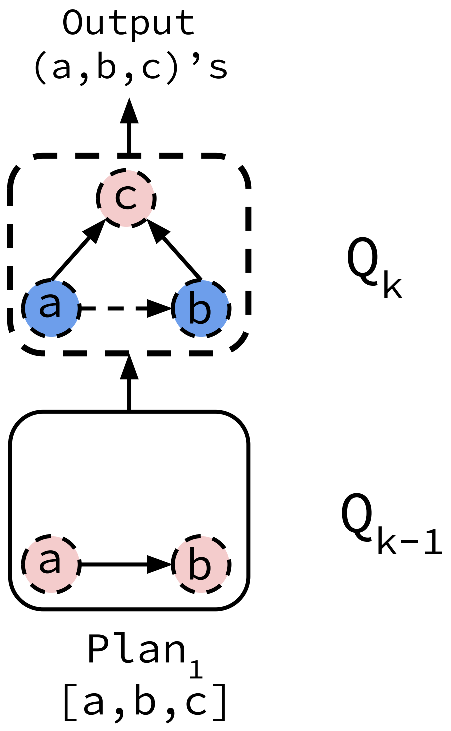
t = (a,b)	
(0, 1)	|fwd(0)| + |fwd(1)|
(0, 2)	
(1, 5)	
...	

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

Within an operator, consider the # input tuples



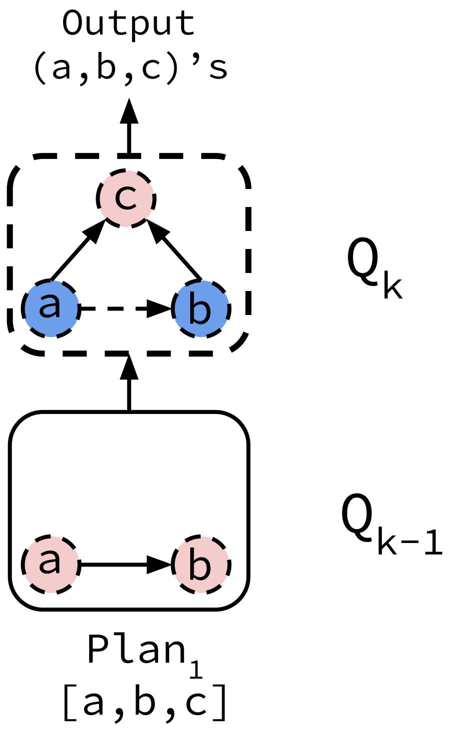
$$\sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. } (i,\, dir) \text{ is accessed}}} |t[i].dir|$$

t = (a,b)	
(0, 1)	|fwd(0)| + |fwd(1)|
🟥 (0, 2)	|fwd(0)| + |fwd(2)|
(1, 5)	
...	

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.
I-cost: size of intersected adj lists throughout execution.

Within an operator, consider the # input tuples

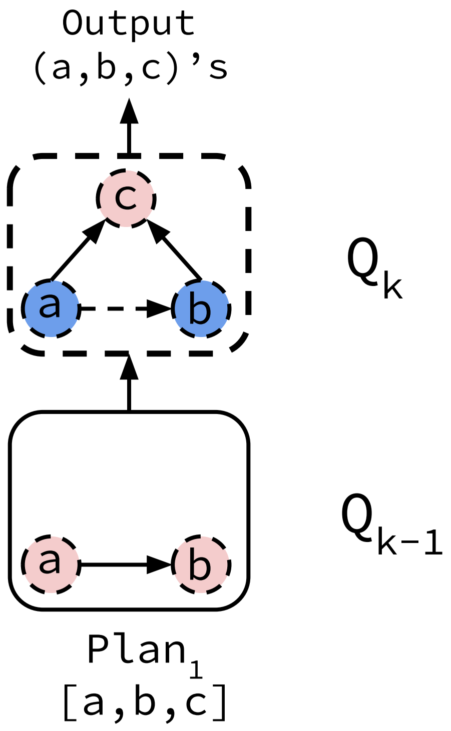


$$\sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. } (i,\, dir) \text{ is accessed}}} |t[i].dir|$$

t = (a,b)	
(0, 1)	|fwd(0)| + |fwd(1)|
(0, 2)	|fwd(0)| + |fwd(2)|
(1, 5)	|fwd(1)| + |fwd(5)|
...	

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.
I-cost: size of intersected adj lists throughout execution.



Within an operator, consider the # input tuples

$$\sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|$$

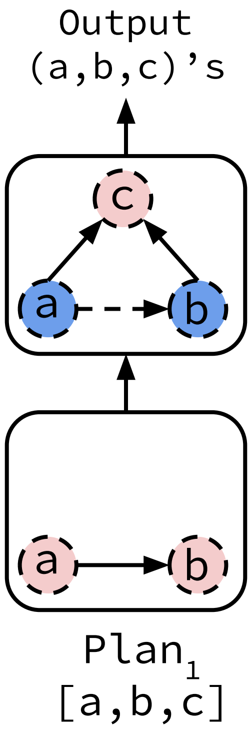
t = (a,b)	
(0, 1)	|fwd(0)| + |fwd(1)| +
(0, 2)	|fwd(0)| + |fwd(2)| +
(1, 5)	|fwd(1)| + |fwd(5)| +
...	...

Operator I-cost

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

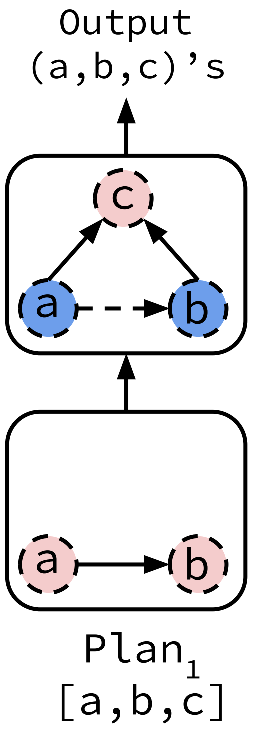


$$\sum_{Q_{k-1} \in Q_2 \ldots Q_{m-1}} \sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|$$

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

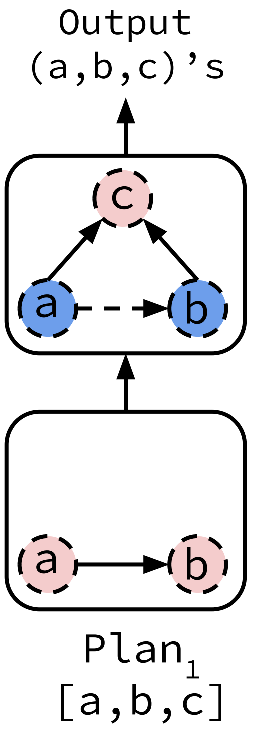


$$\underbrace{\sum_{Q_{k-1} \in Q_2 \ldots Q_{m-1}} \sum_{t \in Q_{k-1}}}_{\textbf{1) } \text{number of intermediate results}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|$$

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

I-cost: size of intersected adj lists throughout execution.

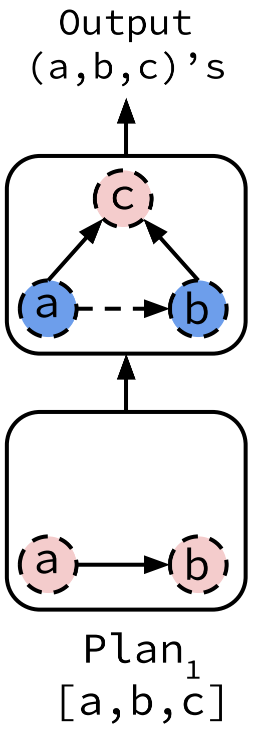


$$\underbrace{\sum_{Q_{k-1} \in Q_2 \dots Q_{m-1}} \sum_{t \in Q_{k-1}}}_{\text{1) number of intermediate results}} \underbrace{\sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|}_{\text{2) size of adj. lists dir. of adj. lists}}$$

Cost Metric - Intersection Cost

Cost of a WCOJ plan is total **intersection-cost** of all operators.

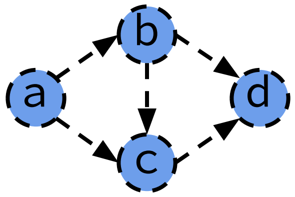
I-cost: size of intersected adj lists throughout execution.



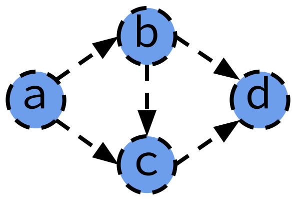
$$\underbrace{\sum_{Q_{k-1} \in Q_2 \ldots Q_{m-1}} \sum_{t \in Q_{k-1}}}_{\textbf{1)}\text{ number of intermediate results}} \underbrace{\sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|}_{\textbf{2)}\text{ size of adj. lists, dir. of adj. lists}}$$

I-Cost captures both effects!
Estimated using a sampling based approach.

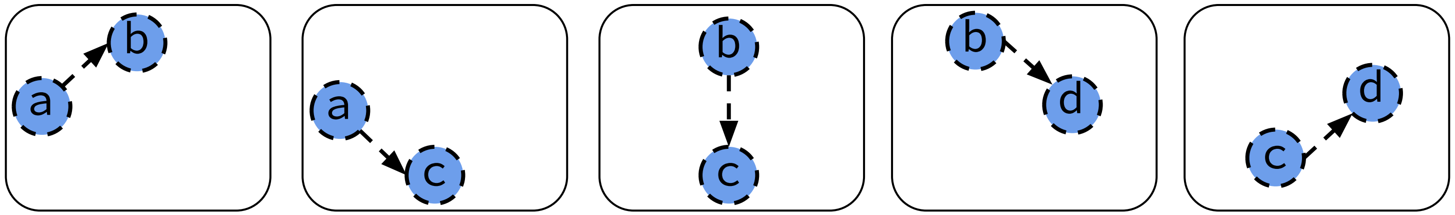
Mixing Binary and Worst-case Optimal Joins



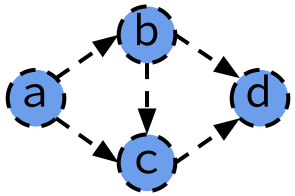
Mixing Binary and Worst-case Optimal Joins



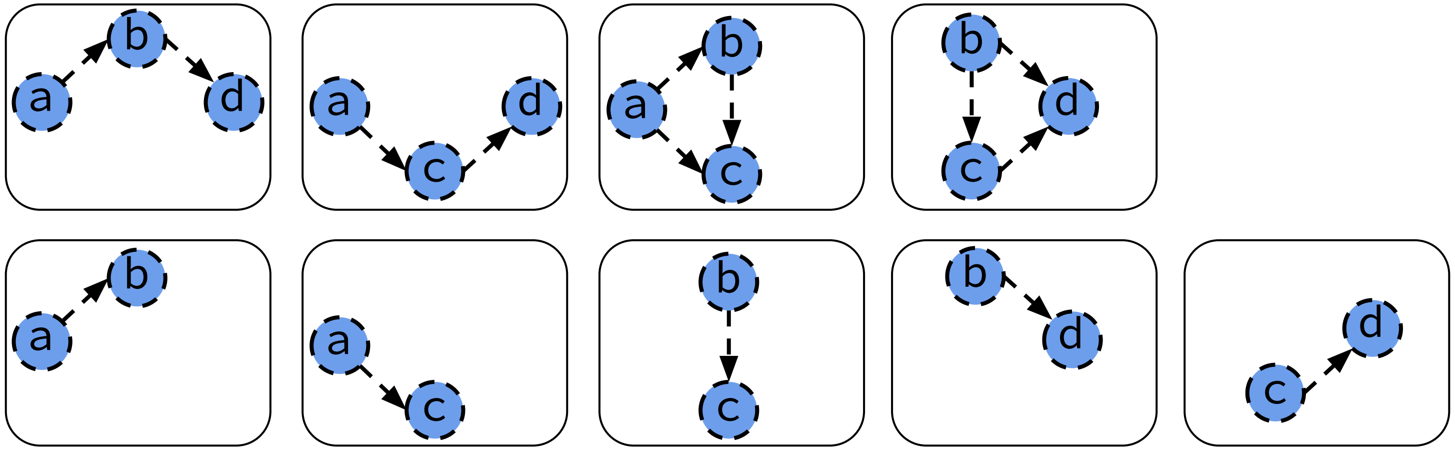
Consider 2-join attributes



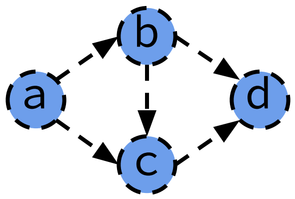
Mixing Binary and Worst-case Optimal Joins



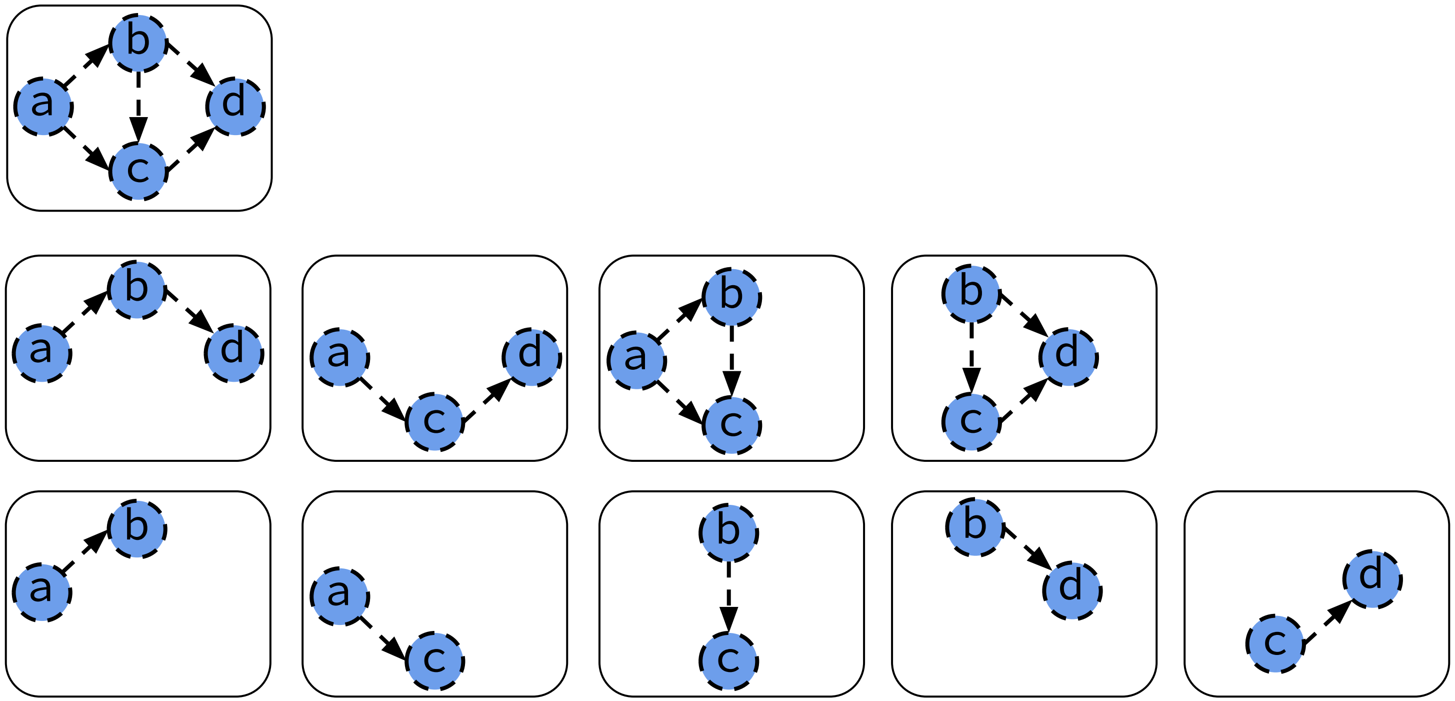
Consider 3-join attributes



Mixing Binary and Worst-case Optimal Joins



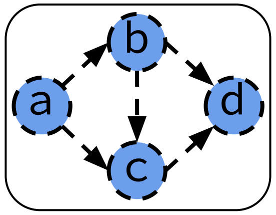
Consider 4-join attributes

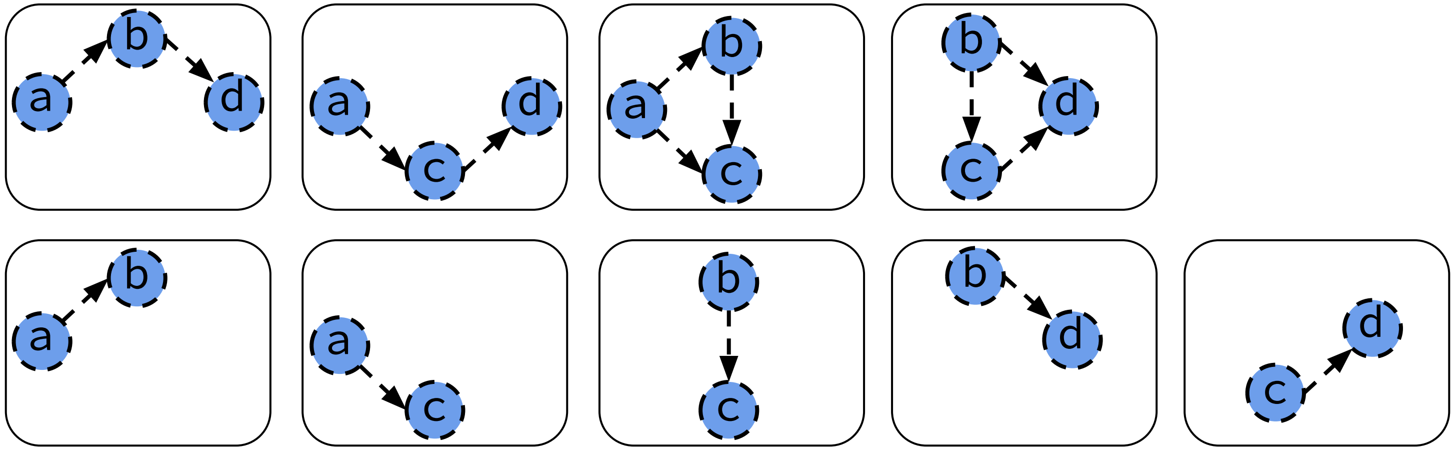


Mixing Binary and Worst-case Optimal Joins

```
At each level:
 ●  Consider the next Join Attribute
 ●  Consider binary joins
```

Consider 4-join attributes

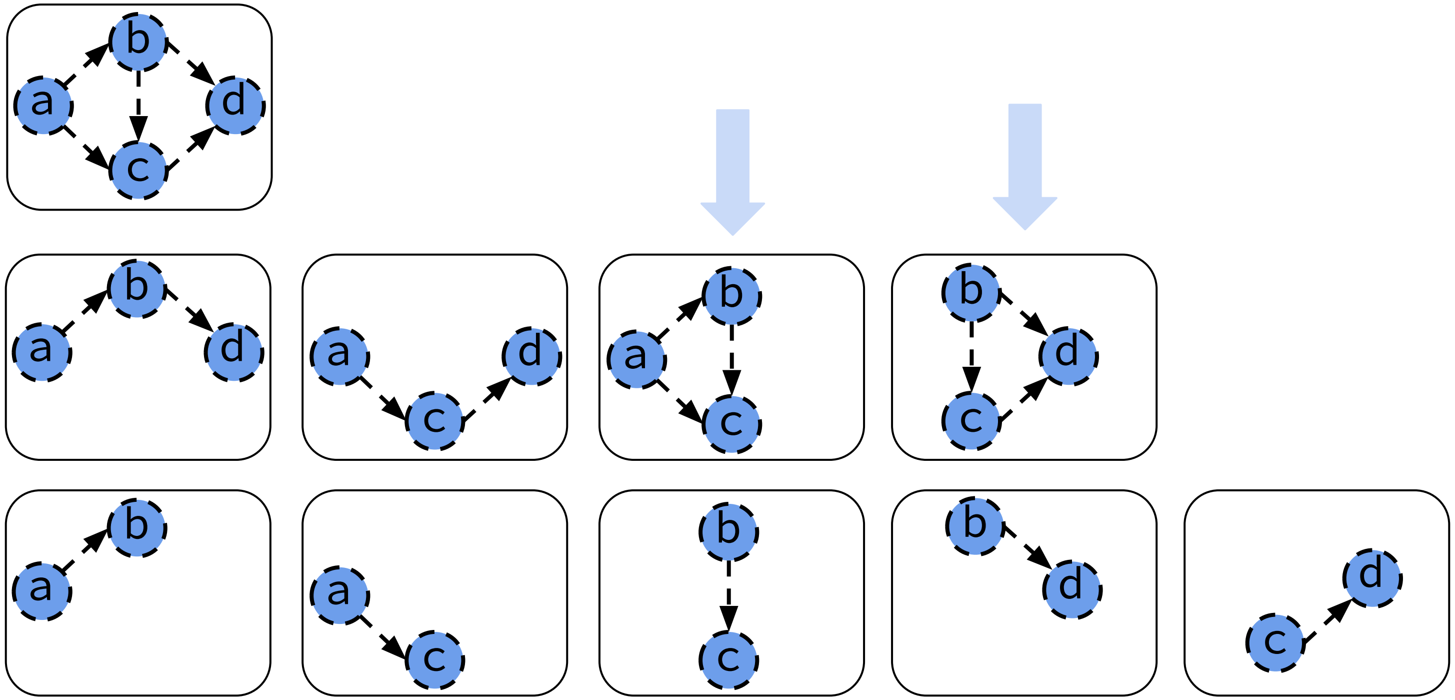




Mixing Binary and Worst-case Optimal Joins

```
At each level:
 ●  Consider the next Join Attribute
 ●  Consider binary joins
```

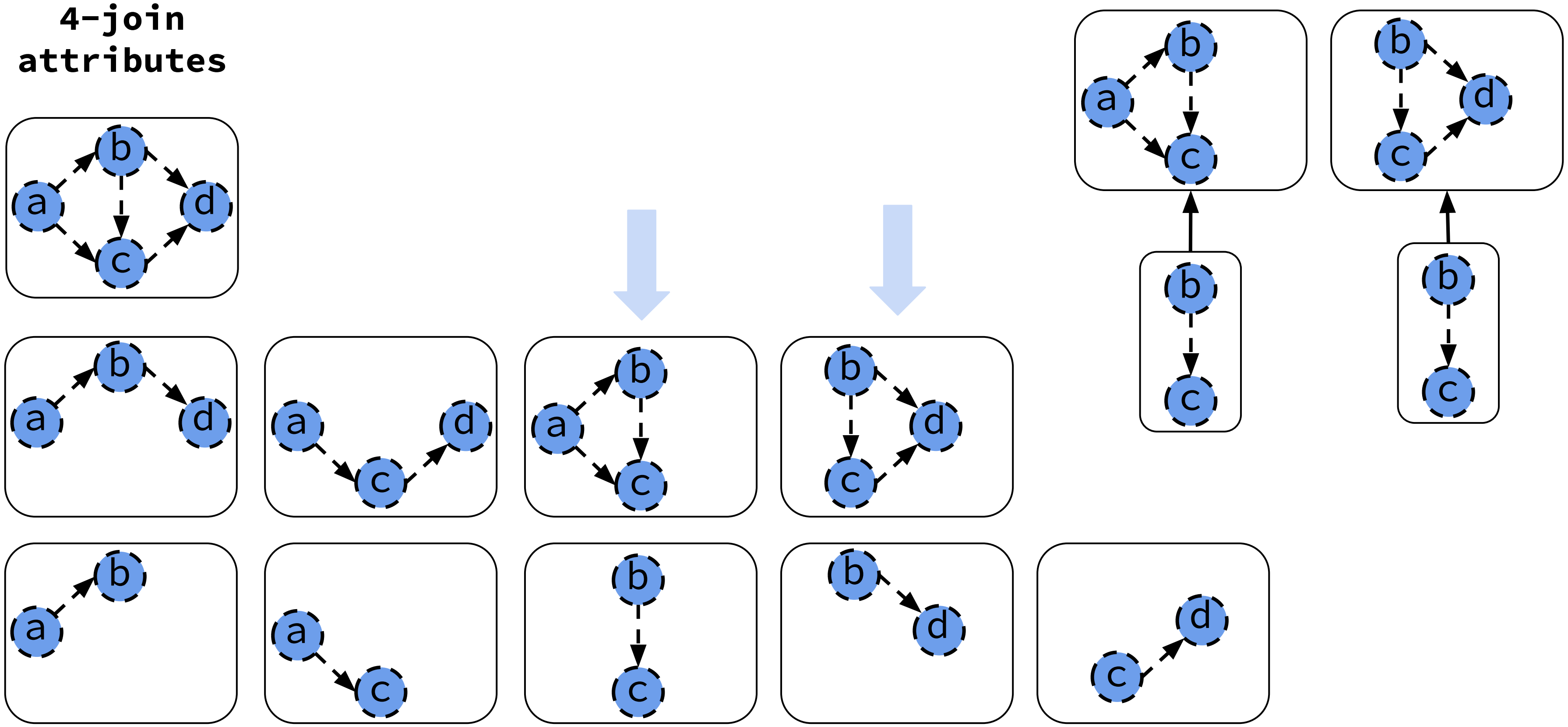
Consider 4-join attributes



Mixing Binary and Worst-case Optimal Joins

```
At each level:
 ●  Consider the next Join Attribute
 ●  Consider binary joins
```

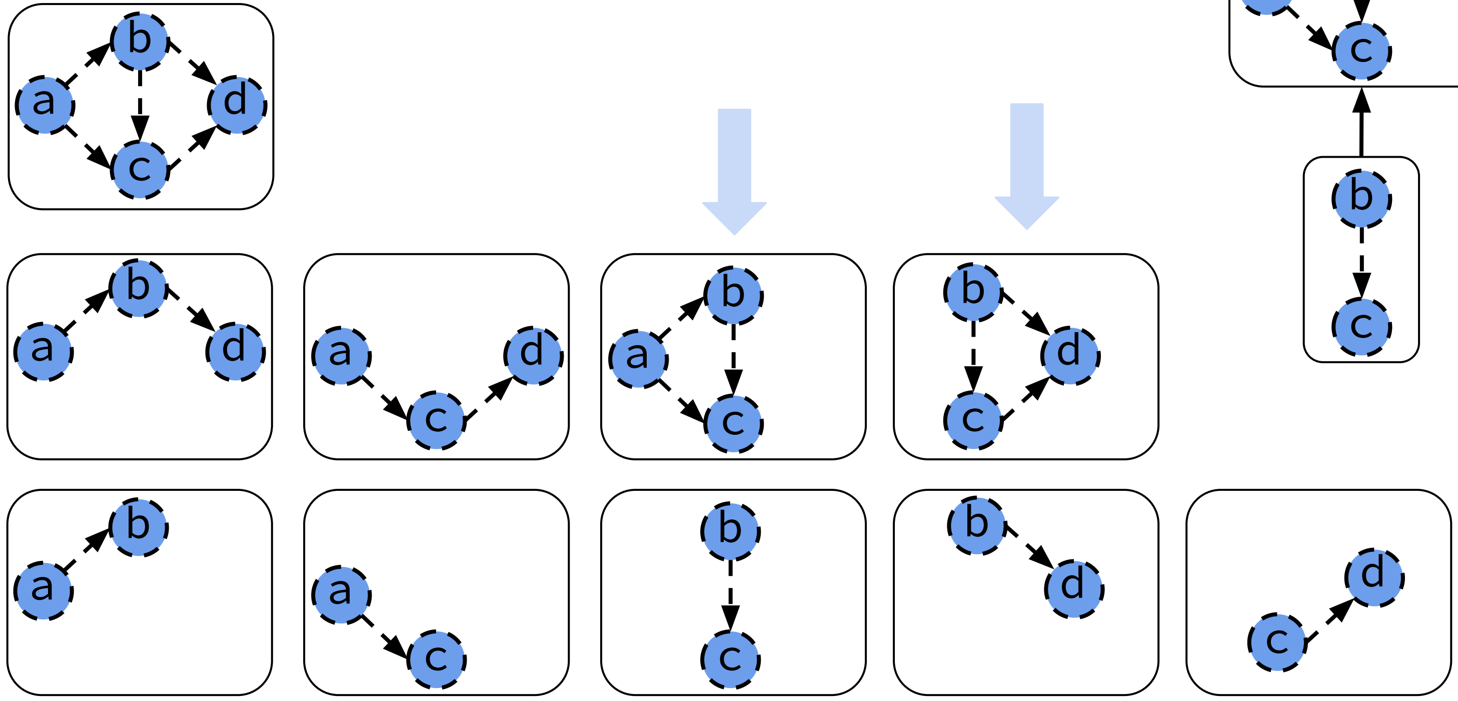
Consider 4-join attributes

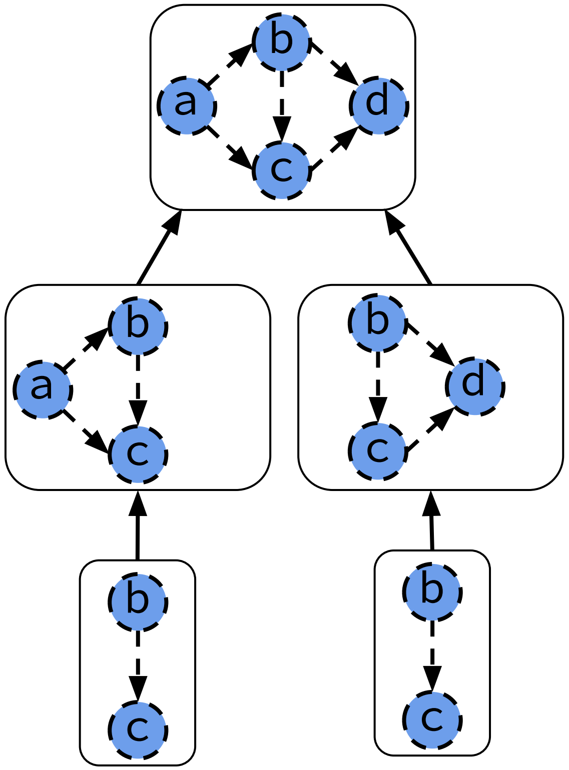


Mixing Binary and Worst-case Optimal Joins

```
At each level:
 •  Consider the next Join Attribute
 •  Consider binary joins
```

Consider 4-join attributes



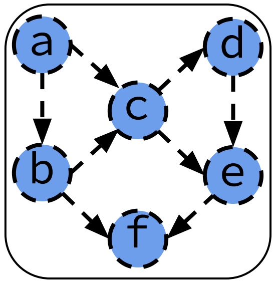


Example Graphflow Hybrid Plan

Dynamic Programming
Query & Dataset
Cost-Based Optimizer

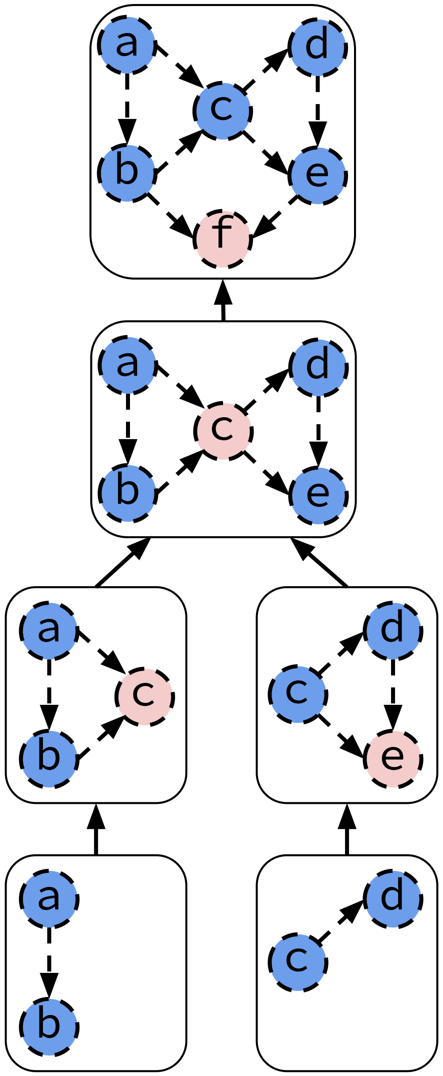
Example Graphflow Hybrid Plan

Dynamic Programming
Query & Dataset
Cost-Based Optimizer



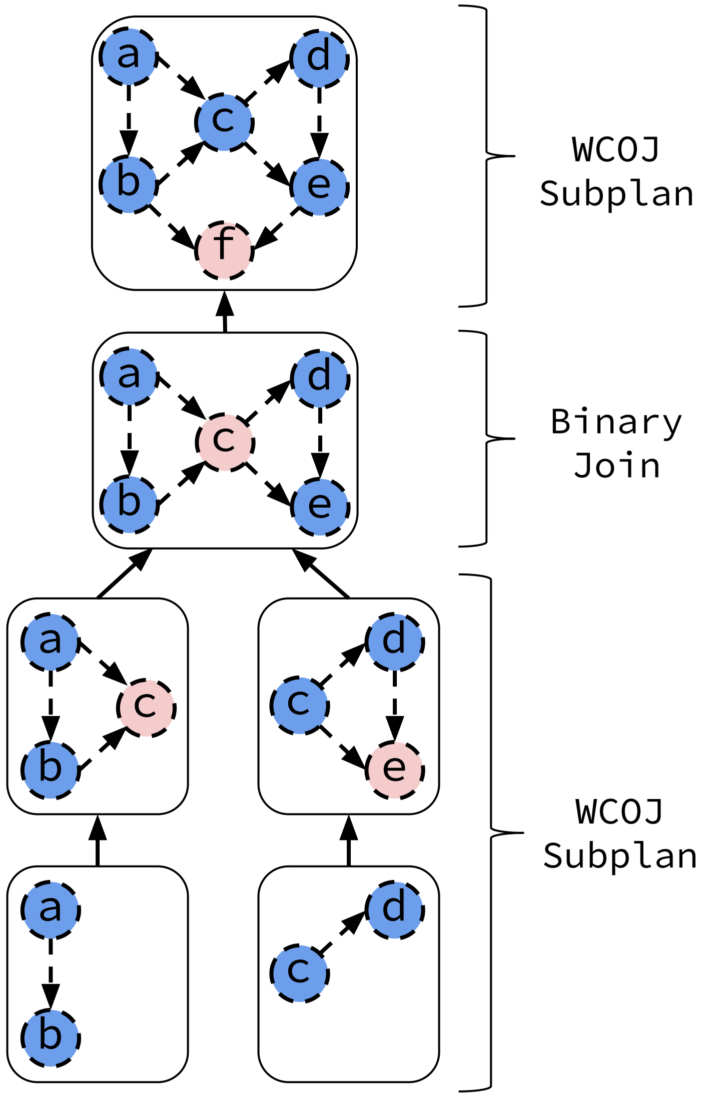
Example Graphflow Hybrid Plan

**Dynamic Programming
Query & Dataset
Cost-Based Optimizer**



Example Graphflow Hybrid Plan

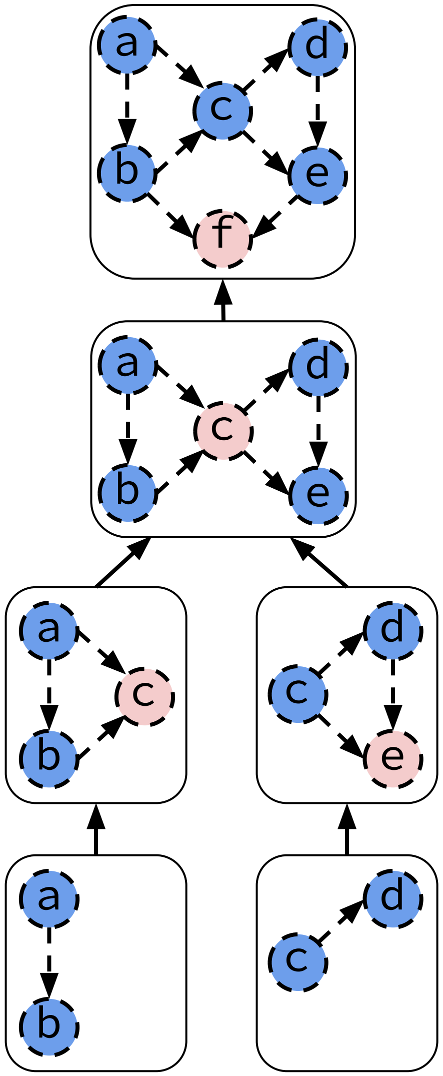
Dynamic Programming
Query & Dataset
Cost-Based Optimizer



Example Graphflow Hybrid Plan

Dynamic Programming Query & Dataset Cost-Based Optimizer

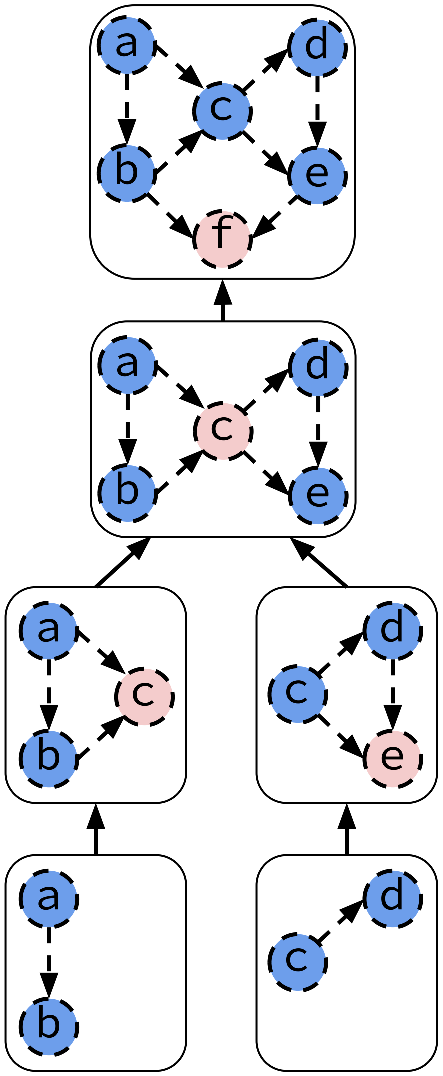
Generalized Hypertree Decomposition Query Cost-Based Optimizer



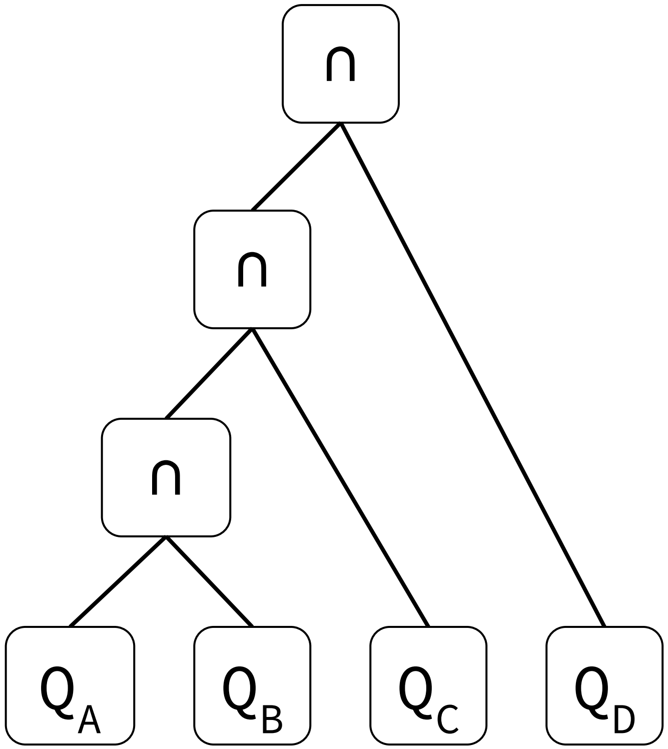
Example Graphflow Hybrid Plan

Dynamic Programming Query & Dataset Cost-Based Optimizer

Generalized Hypertree Decomposition Query Cost-Based Optimizer



This is leads to a specific join plan.



$Q = Q_A \cup Q_B \cup Q_C \cup Q_D$

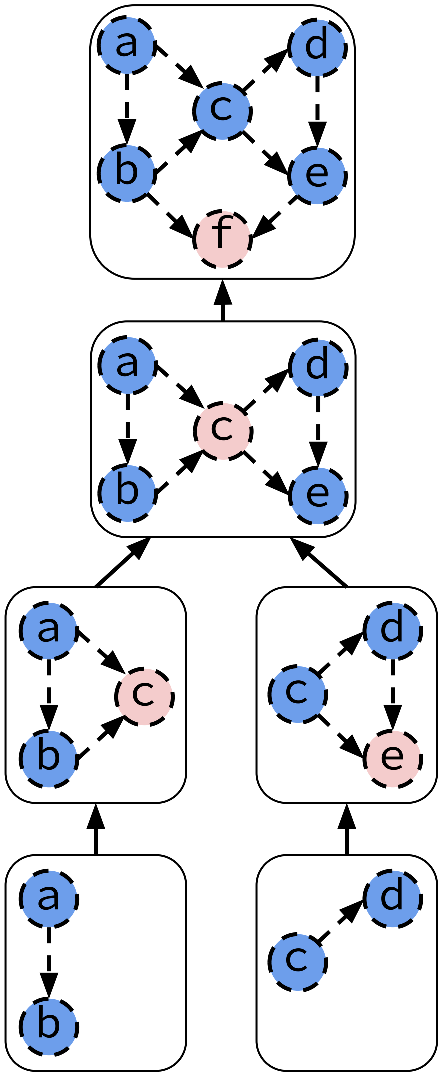
Example Graphflow Hybrid Plan

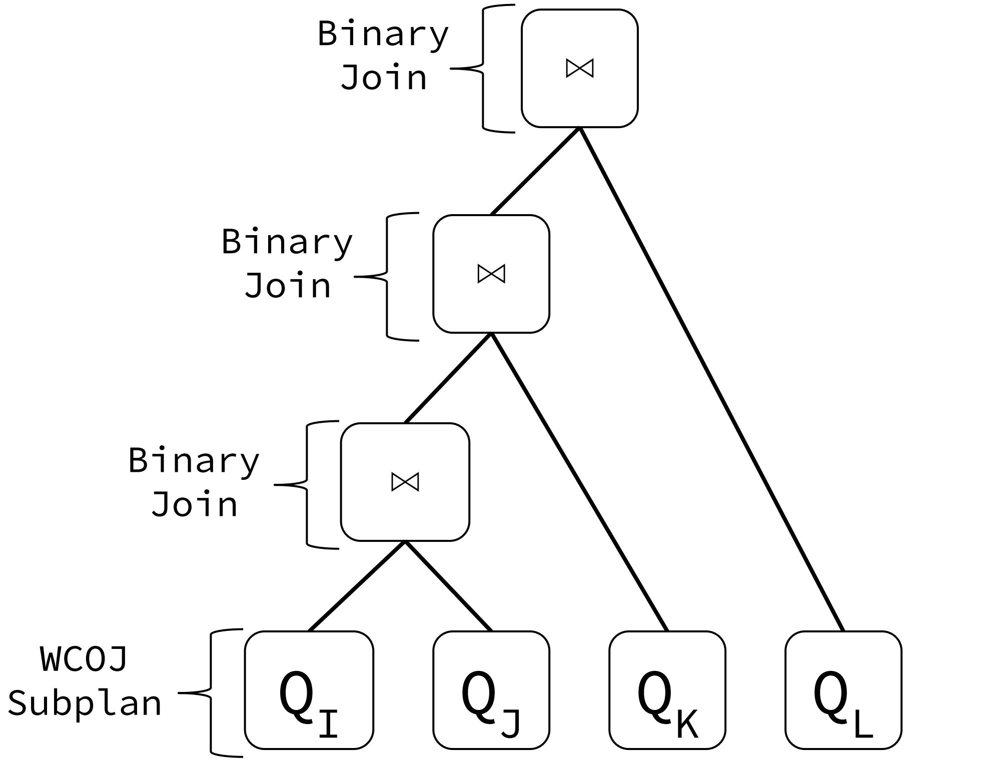
Graphflow

Dynamic Programming Query & Dataset Cost-Based Optimizer

EmptyHeaded

Generalized Hypertree Decomposition Query Cost-Based Optimizer



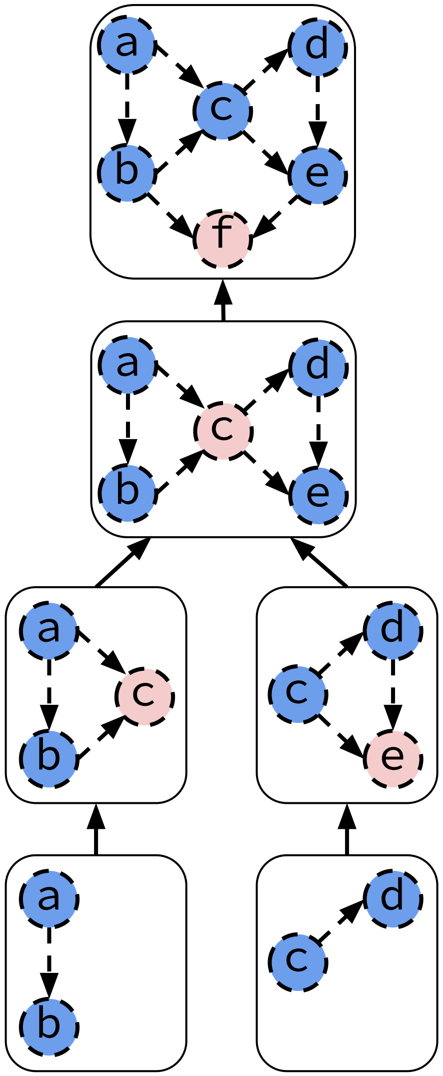


Generates only WCOJ subplans followed by multiple binary joins

Example Graphflow Hybrid Plan

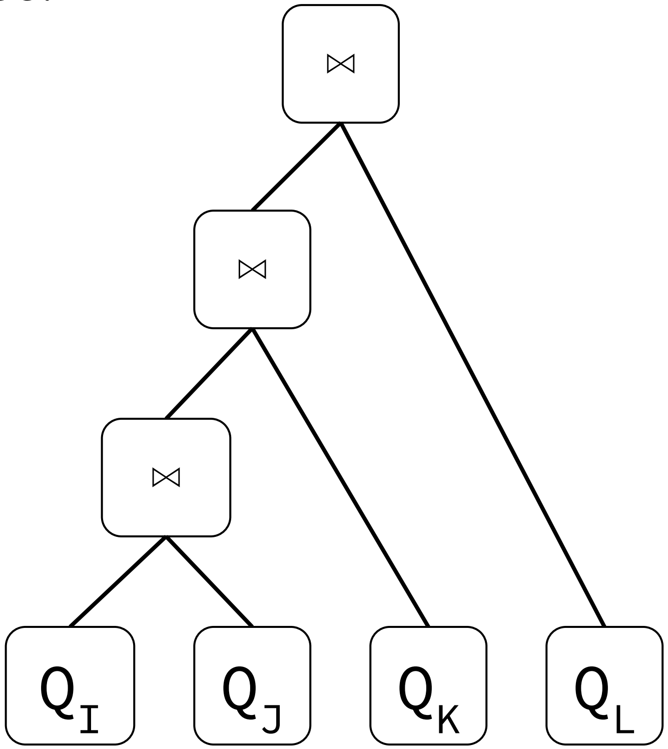
Dynamic Programming Query & Dataset Cost-Based Optimizer

Generalized Hypertree Decomposition Query Cost-Based Optimizer



`Cost:` maximum AGM bound of any of the leaves.

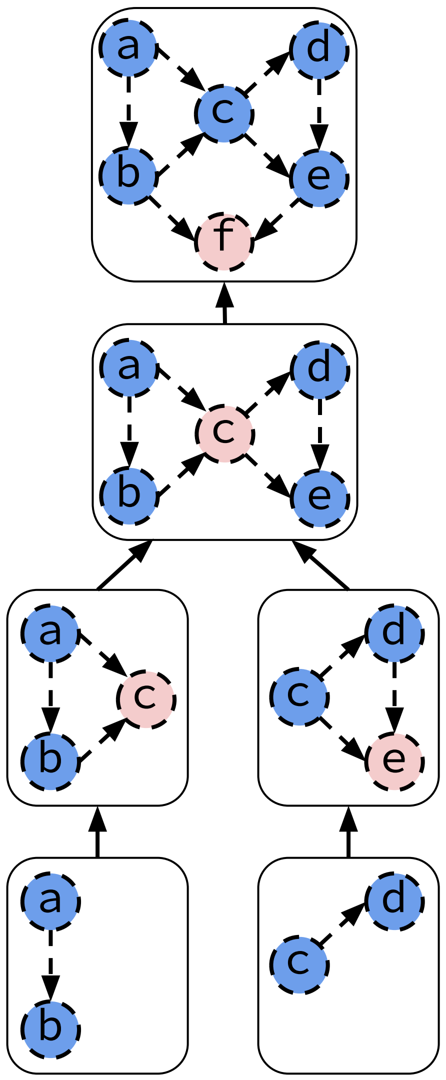
→ Minimize cost!

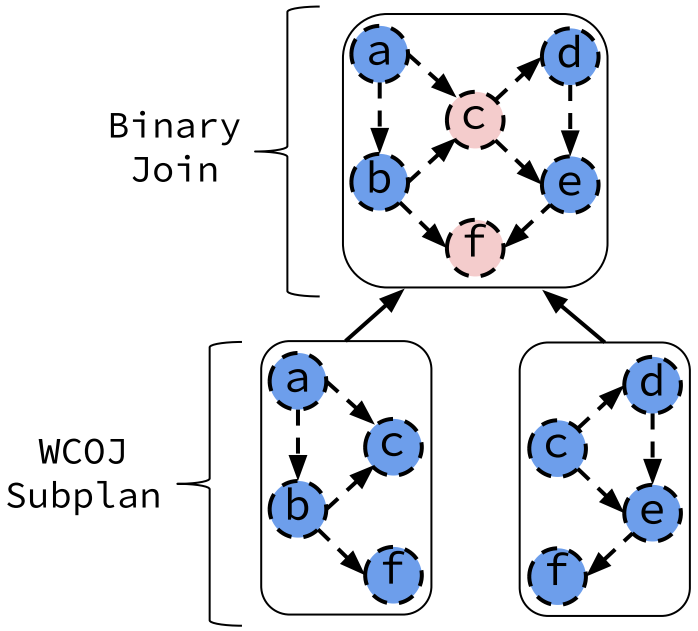


Example Graphflow Hybrid Plan

Dynamic Programming Query & Dataset Cost-Based Optimizer

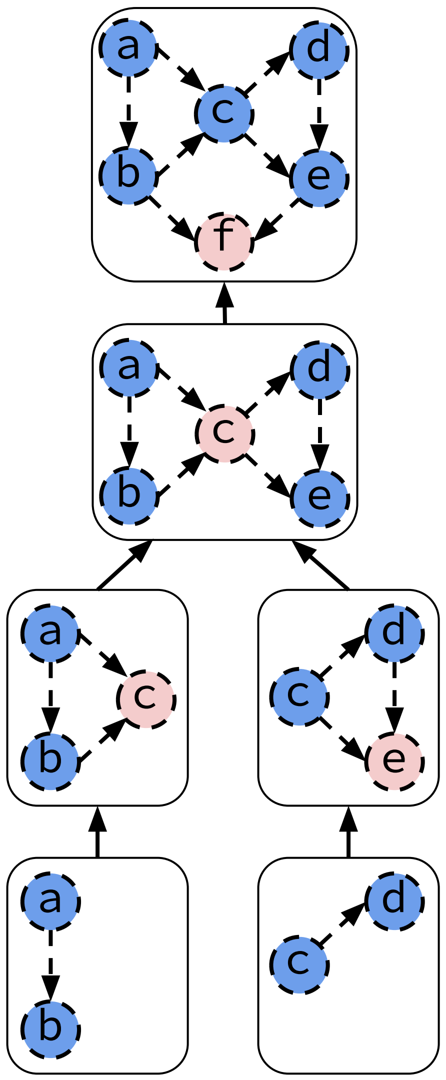
Generalized Hypertree Decomposition Query Cost-Based Optimizer



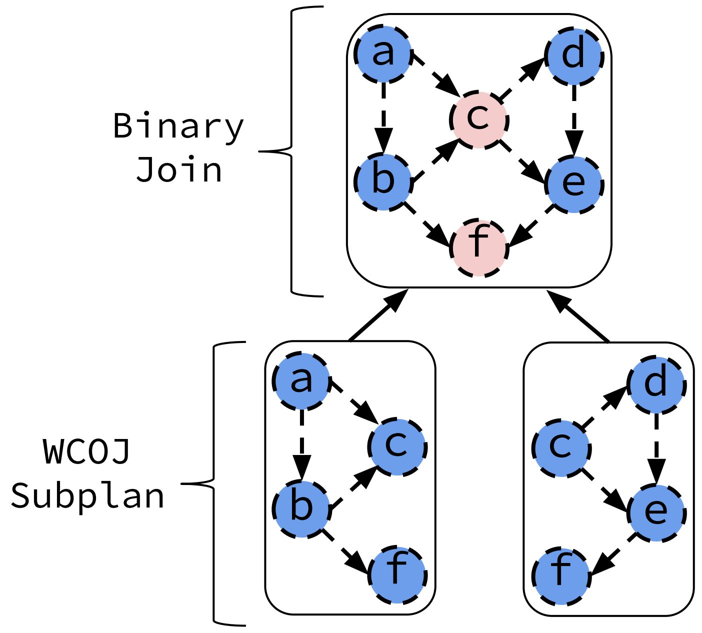


Example Graphflow Hybrid Plan

Dynamic Programming Query & Dataset Cost-Based Optimizer



Generalized Hypertree Decomposition Query Cost-Based Optimizer



Amazon V=403K E=3.4M	**Graphflow**	24.7 secs
	EmptyHeaded (EH)	> 30 mins
	EH in Graphflow	5.8 mins (**14x**)

Graphflow and EmptyHeaded Plan Spaces

Subgraph Queries: 14 queries

Dataset Domains: social networks, web, product co-purchasing

Differ in several structural properties:

(1) size
(2) how skewed their adjacency lists distribution is
(3) average clustering coefficients

Graphflow and EmptyHeaded Comparison

Subgraph Queries: 14 queries

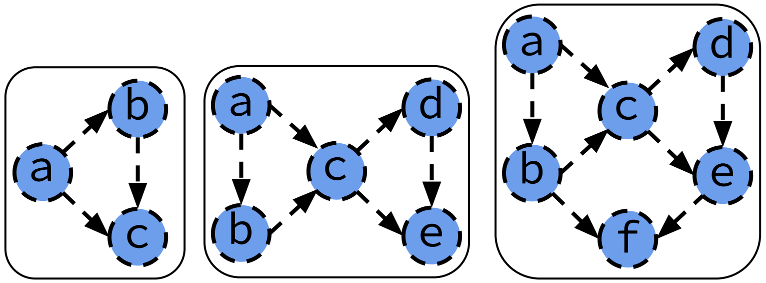
Dataset Domains: social networks, web, product co-purchasing

# Joins	1-3	4-6	7+
Graphflow			
EmptyHeaded + Best JAO orderings			

Graphflow and EmptyHeaded Comparison

Subgraph Queries: 14 queries

Dataset Domains: social networks, web, product co-purchasing



# Joins	1-3	4-6	7+
Graphflow	-	1.8-3.2x Faster 10-25x in rare cases	Runs to completion 4-40 mins **72x speedup**
EmptyHeaded + Best JAO orderings	1.5-2x faster	-	Timeout > 48 hrs

WCOJs Research Impact

WCOJ Adoption

1. Plan Space
2. Cost Model (I-Cost)
3. Cardinality estimator

→ GraphScope

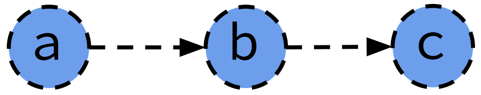
Alibaba

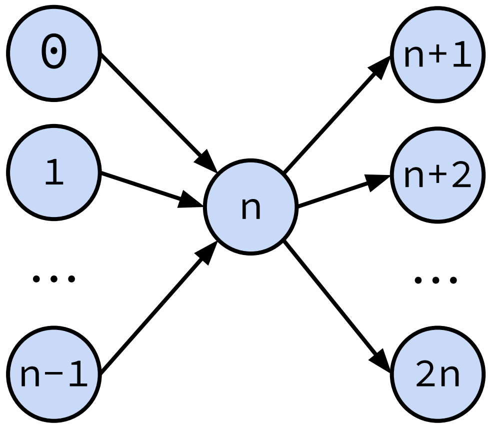
Outline

- **Novel Join Algorithms**
  → Worst-case Optimal Joins

- **Compressed Representations**
  → Factorized Representations

Flat Representation Example

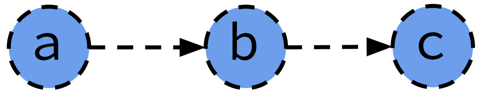


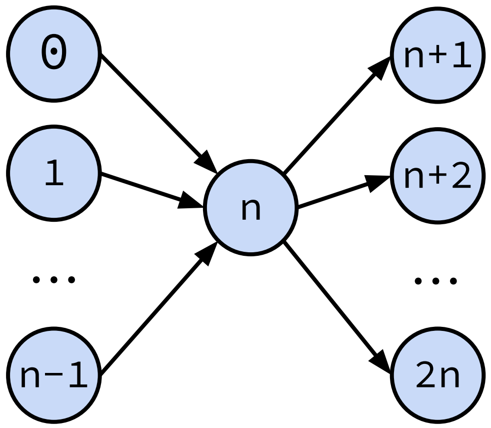


Flat Representation Example

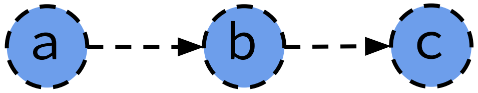
Join Attribute Ordering (JAO): [b,a,c]

```
( b,  a,  c)
```

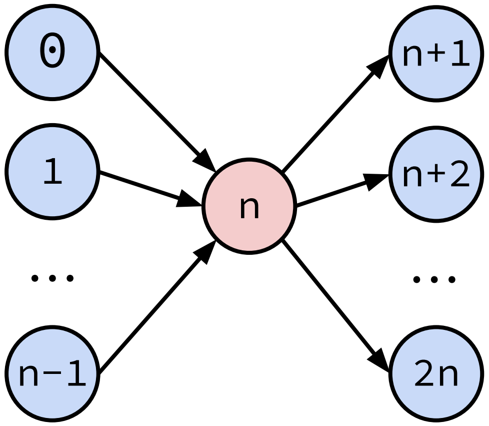




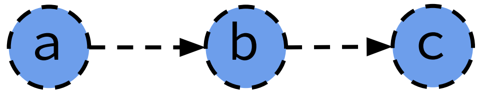
Flat Representation Example



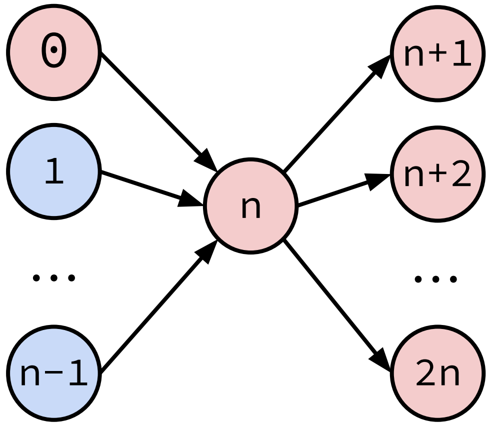
```
( b,  a,  c)
```



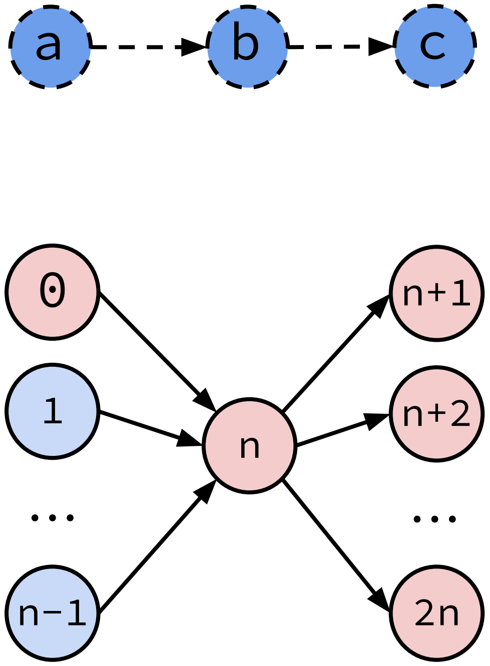
Flat Representation Example



```
( b,  a,  c)
```



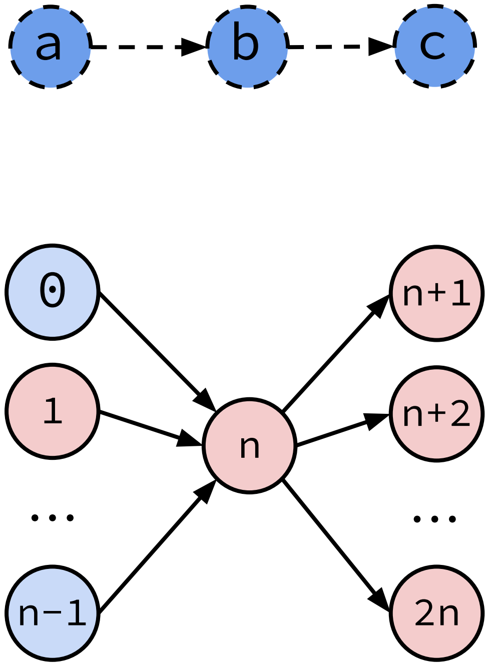
Flat Representation Example



(b,	a,	c)
(n,	0,	n+1)
(n,	0,	...)
(n,	0,	2n)

$\}$ **n tuples**

Flat Representation Example

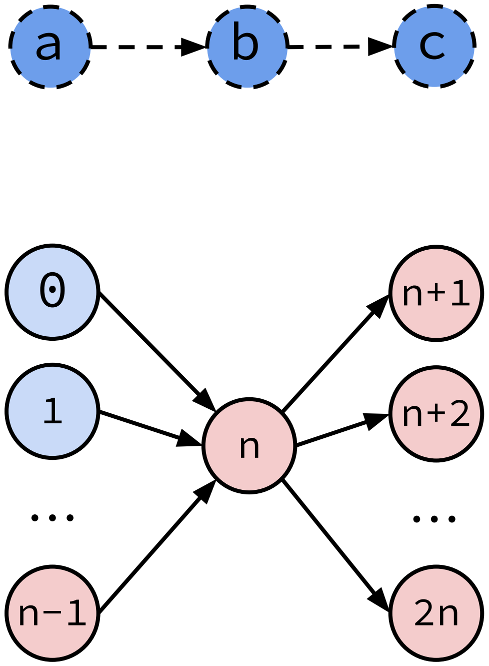


(b,	a,	c)
(n,	0,	n+1)
(n,	0,	...)
(n,	0,	2n)
(n,	1,	n+1)
(n,	1,	...)
(n,	1,	2n)

$\left.\begin{array}{l}(n, 0, n+1)\\(n, 0, \ldots)\\(n, 0, 2n)\end{array}\right\}$ **n tuples**

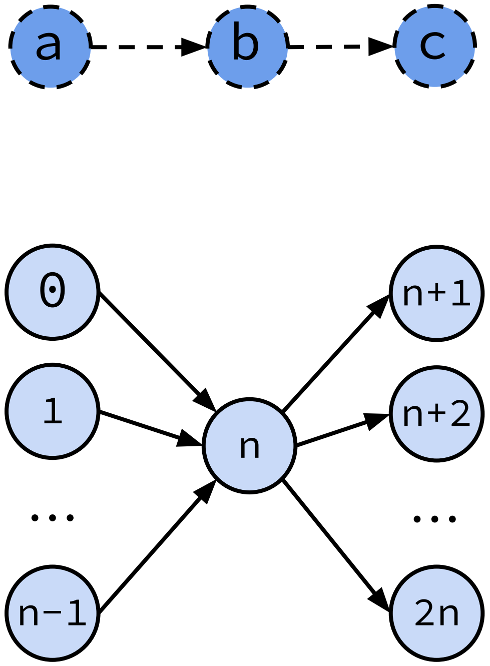
$\left.\begin{array}{l}(n, 1, n+1)\\(n, 1, \ldots)\\(n, 1, 2n)\end{array}\right\}$ **n tuples**

Flat Representation Example



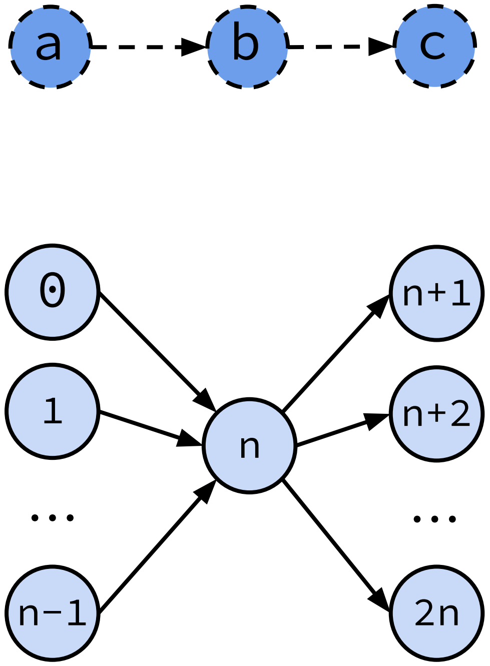
```
( b,   a,   c)
--------------
( n,   0, n+1)  ⎫
( n,   0, ...)  ⎬ n tuples
( n,   0, 2n)   ⎭
( n,   1, n+1)  ⎫
( n,   1, ...)  ⎬ n tuples
( n,   1, 2n)   ⎭
        ...
( n, n-1, n+1)  ⎫
( n, n-1,  ...) ⎬ n tuples
( n, n-1,  2n)  ⎭
```

Flat Representation Example



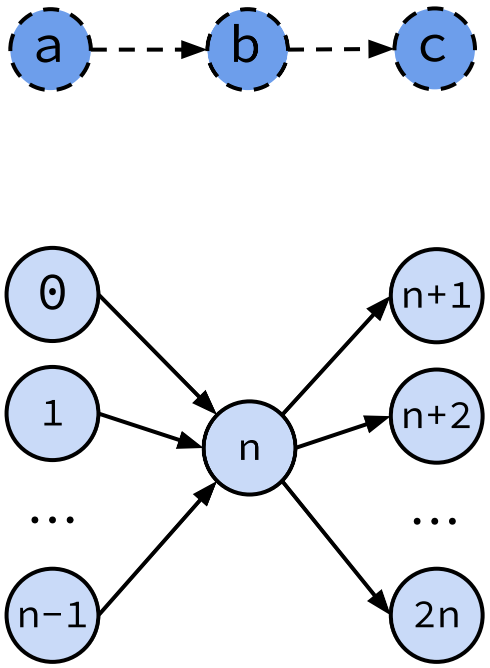
```
( b,  a,  c)
-------------
( n,  0, n+1)   }
( n,  0, ...)   }  n tuples
( n,  0, 2n)    }
( n,  1, n+1)   }
( n,  1, ...)   }  n tuples
( n,  1, 2n)    }
      ...
( n, n-1, n+1)  }
( n, n-1,  ...) }  n tuples
( n, n-1,  2n)  }
```

Flat Representation Example



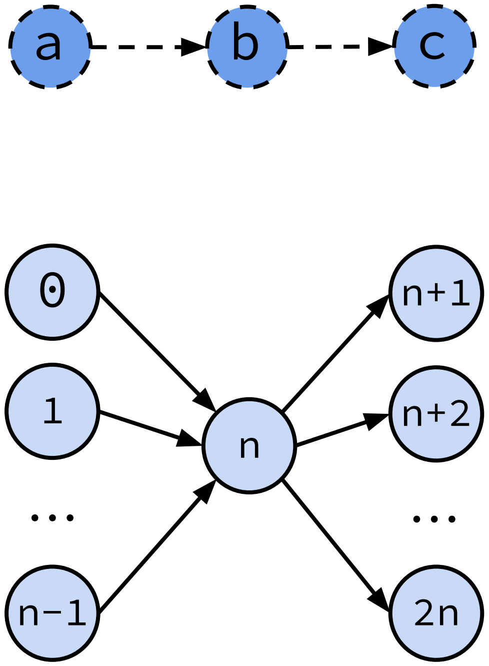
```
( b,   a,   c)
--------------
( n,   0, n+1)   }
( n,   0, ...)   } n tuples
( n,   0,  2n)   }
( n,   1, n+1)   }
( n,   1, ...)   } n tuples
( n,   1,  2n)   }
       ...
( n, n-1, n+1)   }
( n, n-1,  ...)  } n tuples
( n, n-1,   2n)  }
```

Flat Representation Example



```
( b,  a,  c)
--------------
( n,  0, n+1)   \
( n,  0, ...)    }  n tuples
( n,  0, 2n)    /
( n,  1, n+1)   \
( n,  1, ...)    }  n tuples
( n,  1, 2n)    /
      ...
  n, n-1  n+1)  \
  n, n-1   ...)  }  n tuples
  n, n-1   2n)  /
```

Flat Representation Example

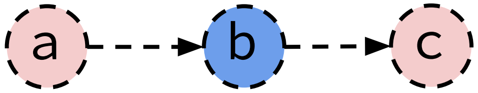


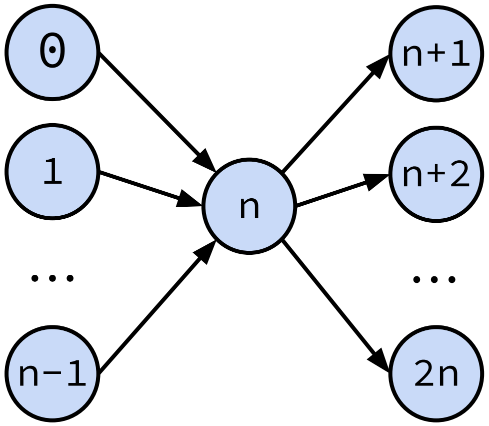
b	a	c
n	0	n+1
n	0	...
n	0	2n
n	1	n+1
n	1	...
n	1	2n
	...	
n	n-1	n+1
n	n-1	...
n	n-1	2n

n^2 tuples

Conditional Independence

a and c are
conditionally independent on b!!

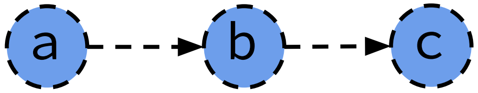


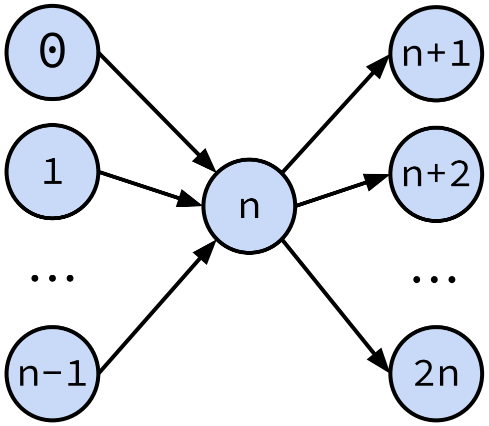


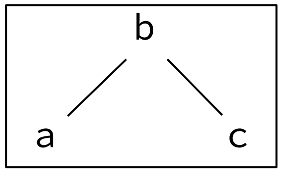
(b,	**a,**	**c)**
(n,	0,	n+1)
(n,	0,	...)
(n,	0,	2n)
(n,	1,	n+1)
(n,	1,	...)
(n,	1,	2n)
	...	
(n,	n-1,	n+1)
(n,	n-1,	...)
(n,	n-1,	2n)

$\}$ **n^2 tuples**

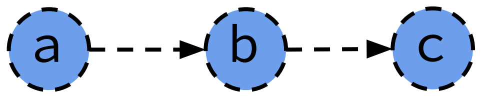
Factorized Representation Example

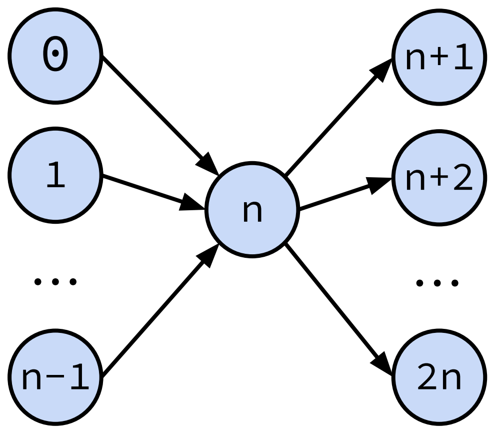


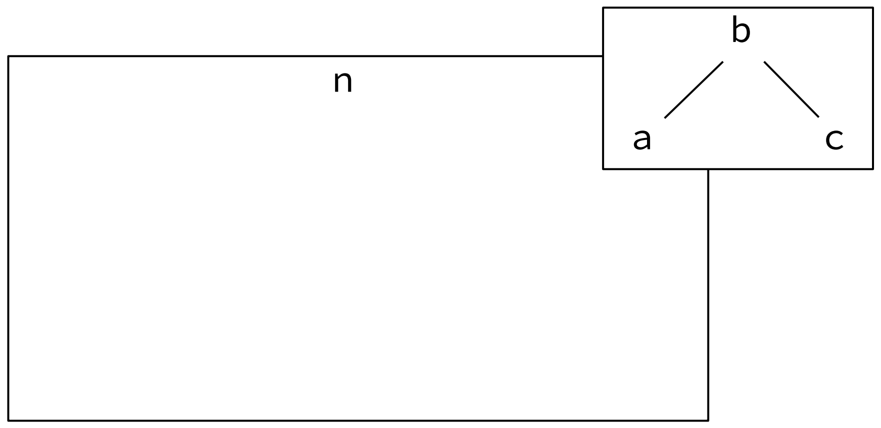




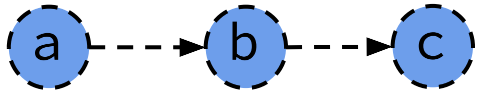
Factorized Representation Example

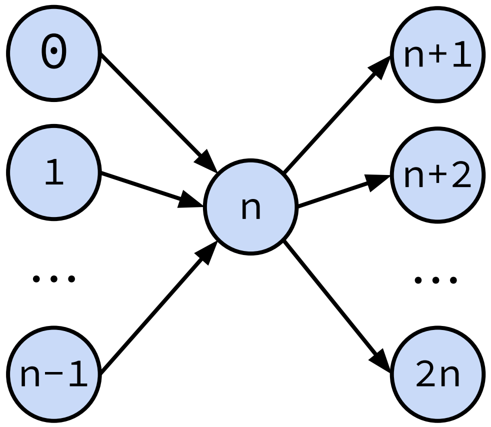


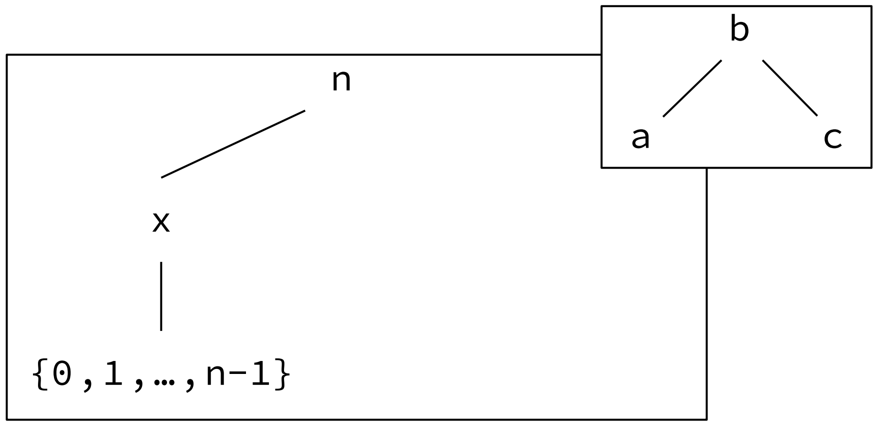




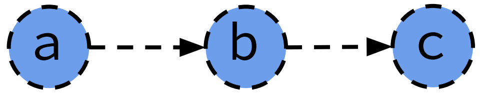
Factorized Representation Example

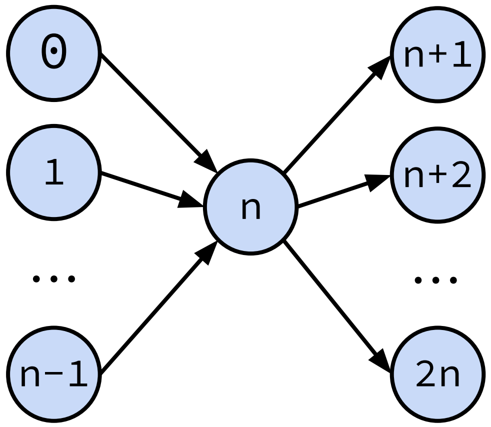


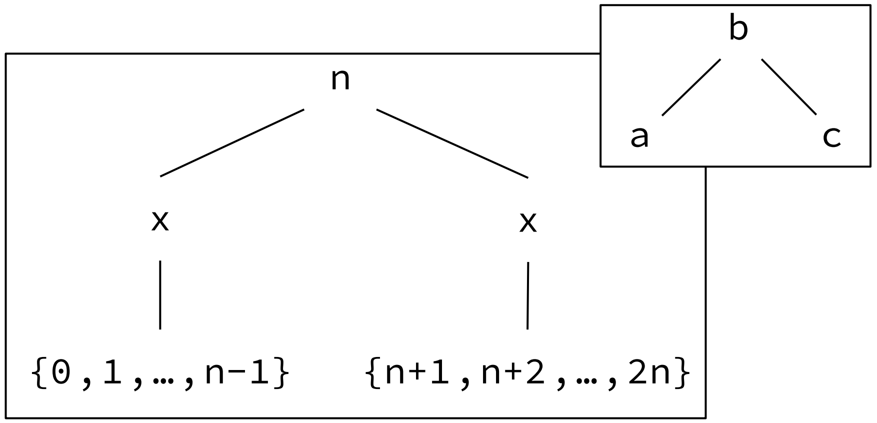




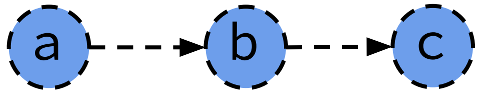
Factorized Representation Example

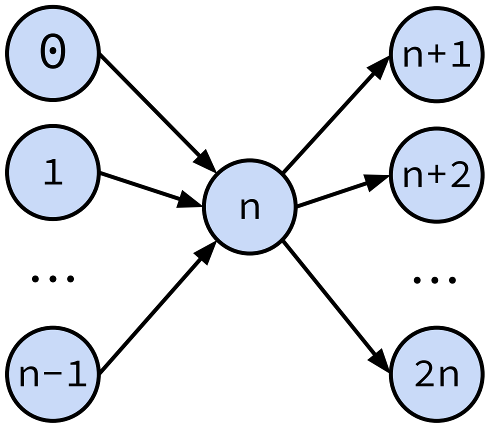


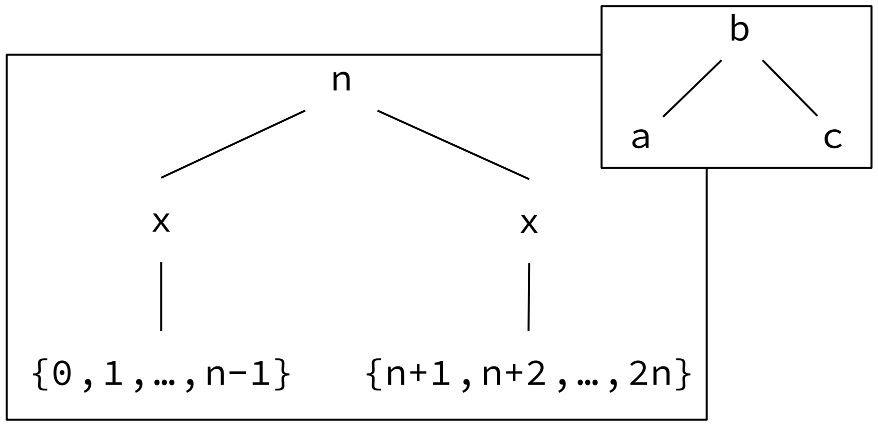




Factorized Representation Example



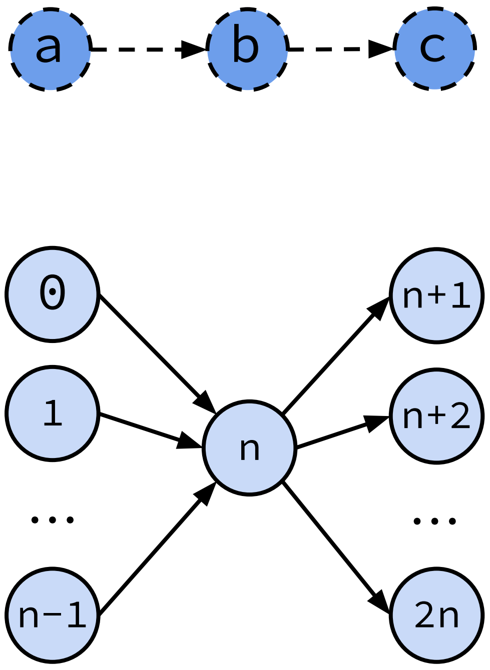


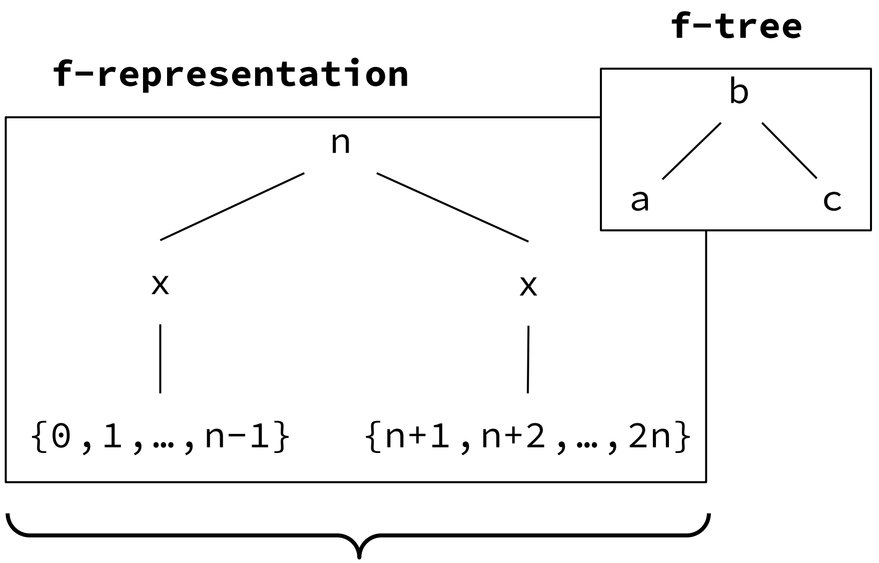


$\underbrace{\hphantom{xxxxxxxxxxxxxxxxxxxxxxxx}}$

2n+1 fields
vs.
$3n^2$ fields (flat)

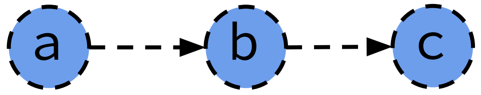
Factorized Representation Example

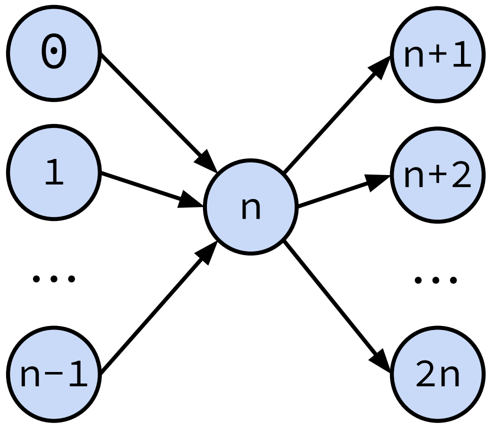




2n+1 fields
vs.
$3n^2$ fields (flat)

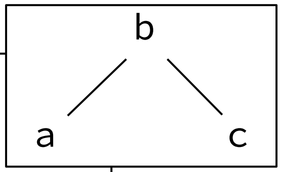
Theory on F-Representations



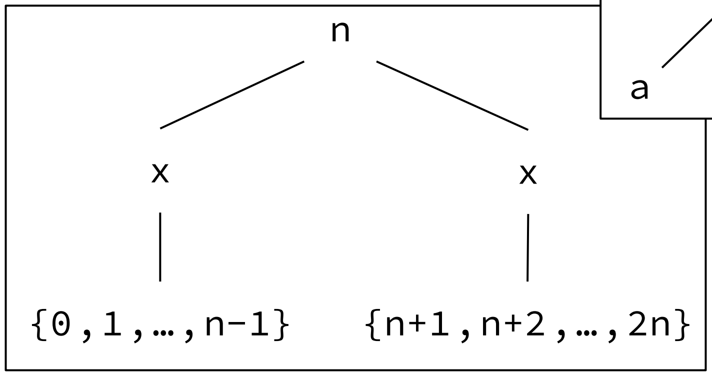


```
Theory of factorization:
          σ(Q) ≤ AGM(Q)

where
σ(Q): worst-case size bound
      over f-representations.

In some cases,
         σ(Q) <  AGM(Q)
```

f-tree



f-representation



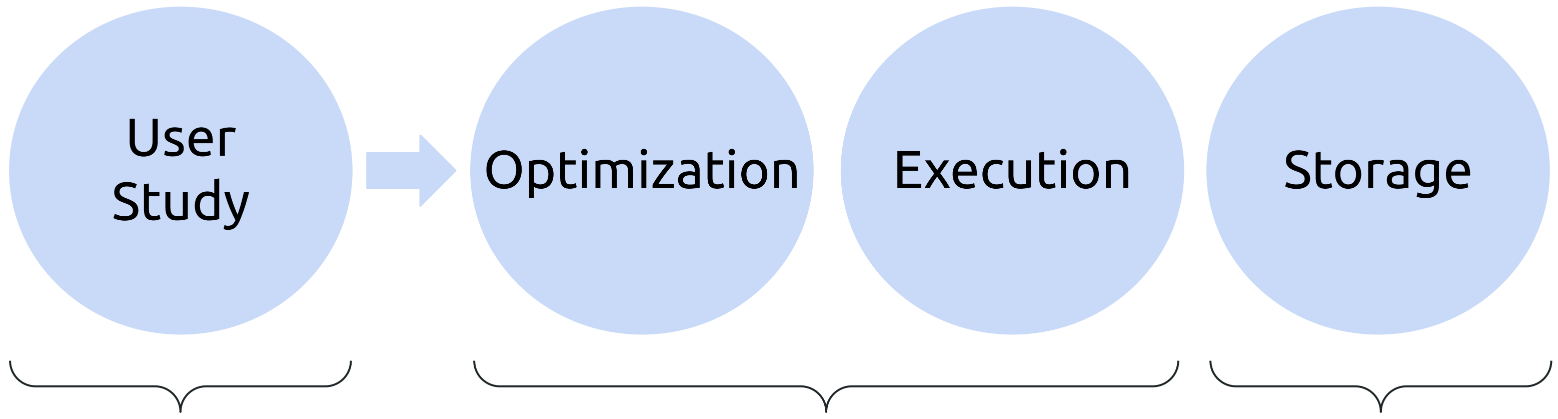
$\underbrace{\hspace{8cm}}$

2n+1 fields

vs.

$3n^2$ fields (flat)

Fin.
Questions?



Best Paper Award
VLDB 2018
VLDBJ 2020

Novel Join Algorithms
VLDB 2019, TODS 2021
Compressed Representations
VLDB 2021

Specialized Graph-Native
ICDE 2021
VLDB 2021

Graph*flow*

SIGMOD 2017