Query Evaluation Revised: Parallel, Distributed, via Rewritings

Doctoral Defence

Christopher Spinrath

TU Dortmund University

January 29, 2024

Query Evaluation

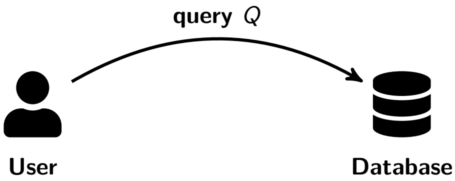
Classical Query Evaluation (simplified)

User

Database

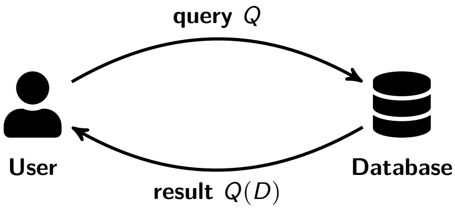
Query Evaluation

Classical Query Evaluation (simplified)



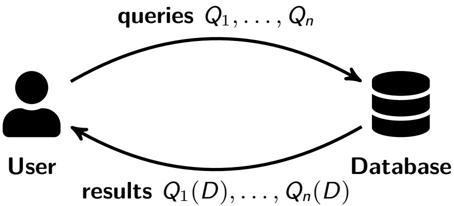
Query Evaluation

Classical Query Evaluation (simplified)



Query Evaluation

Query Evaluation (simplified)

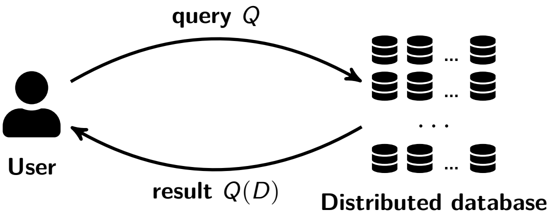

queries $Q_1, \ldots, Q_n$
User
Database
results $Q_1(D), \ldots, Q_n(D)$

Examples for Circumstances

- Multiple input queries

Query Evaluation

Query Evaluation (simplified)

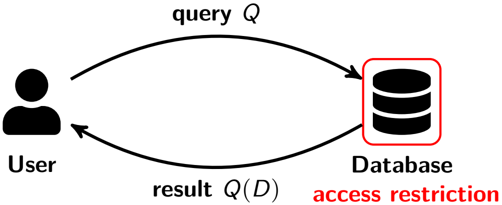


Examples for Circumstances

- Multiple input queries
- Distributed database(s)

Query Evaluation

Query Evaluation (simplified)

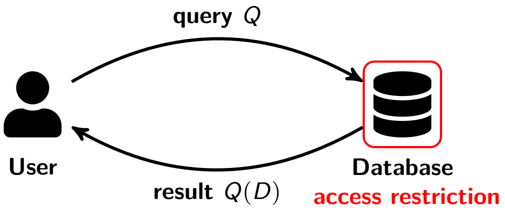


Examples for Circumstances

- Multiple input queries
- Distributed database(s)
- Access Restrictions

Query Evaluation

Query Evaluation (simplified)

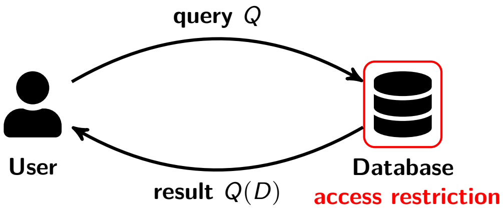


Examples for Circumstances

- Multiple input queries
- Distributed database(s)
- Access Restrictions
- ...

Query Evaluation

Query Evaluation (simplified)



Examples for Circumstances

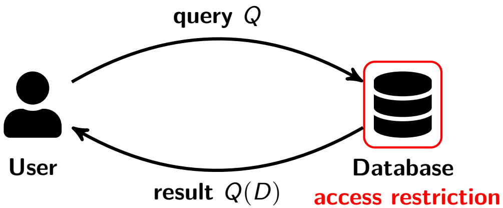
- Multiple input queries
- Distributed database(s)
- Access Restrictions
- ...

Questions

- Are methods from the classical setting still suitable?
  - Algorithms, Correctness, Complexity, ...

Query Evaluation

Query Evaluation (simplified)



Examples for Circumstances

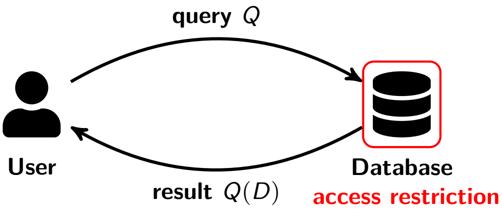
- Multiple input queries
- Distributed database(s)
- Access Restrictions
- ...

Questions

- Are methods from the classical setting still suitable?
  - Algorithms, Correctness, Complexity, ...
- What is considered “suitable”?

Query Evaluation

Query Evaluation (simplified)



Examples for Circumstances

- Multiple input queries
- Distributed database(s)
- Access Restrictions
- ...

Questions

- Are methods from the classical setting still suitable?
  - Algorithms, Correctness, Complexity, ...
- What is considered “suitable”?
- (How) can methods be adapted?

Settings

1. Work-Efficient Constant-Time Parallel Query Evaluation

---

2. Parallel-Correctness and -Boundedness of Datalog Queries

---

3. Structurally Simple Rewritings

Settings

1. Work-Efficient Constant-Time Parallel Query Evaluation

data complexity

2. Parallel-Correctness and -Boundedness of Datalog Queries

static analysis

3. Structurally Simple Rewritings

static analysis

Settings

1. Work-Efficient Constant-Time Parallel Query Evaluation
   Preliminary results published at ICDT'23, Ioannina, Greece
   (Keppeler, Schwentick, and **S.** 2023)

data complexity

---

2. Parallel-Correctness and -Boundedness of Datalog Queries

static analysis

---

3. Structurally Simple Rewritings

static analysis

Query Evaluation

$$Q\colon R \bowtie S$$

relational algebra query

Query Evaluation

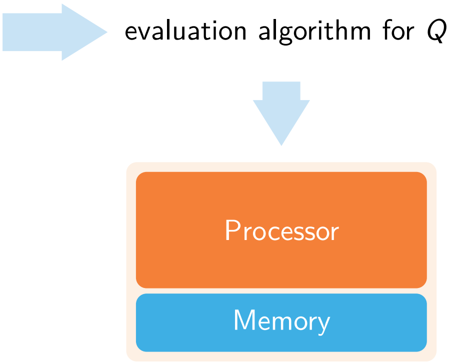
$Q\colon R \bowtie S$ → evaluation algorithm for Q

relational algebra query

Query Evaluation

$Q\colon R \bowtie S$

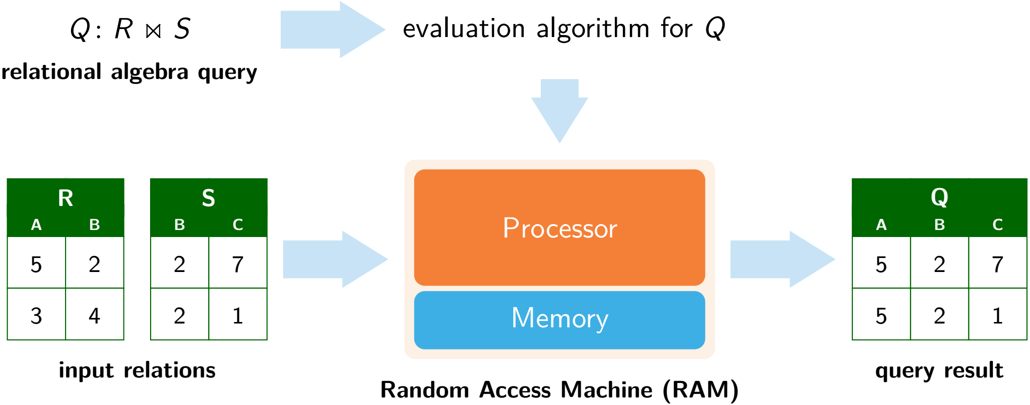
relational algebra query



evaluation algorithm for Q

Random Access Machine (RAM)

Query Evaluation



$Q\colon R \bowtie S$

relational algebra query

evaluation algorithm for Q

R	
A	B
5	2
3	4

S	
B	C
2	7
2	1

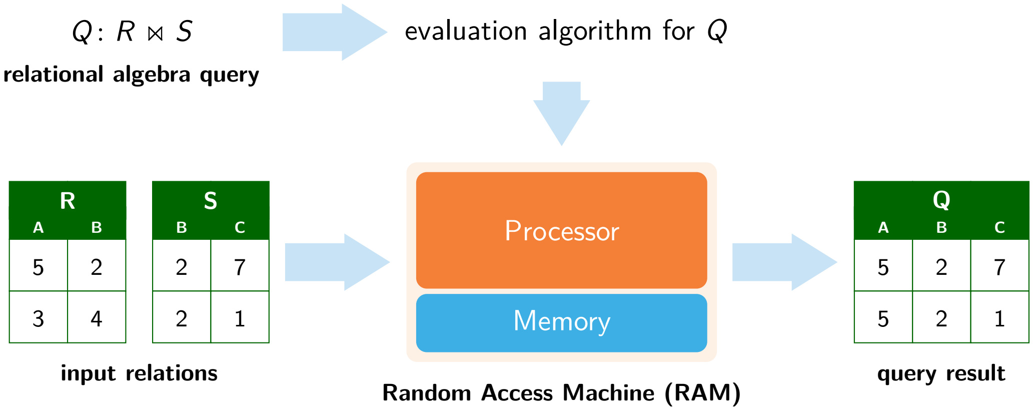
input relations

Random Access Machine (RAM)

Q		
A	B	C
5	2	7
5	2	1

query result

Query Evaluation



$Q\colon R \bowtie S$

relational algebra query

evaluation algorithm for Q

R	
A	B
5	2
3	4

S	
B	C
2	7
2	1

input relations

Processor

Memory

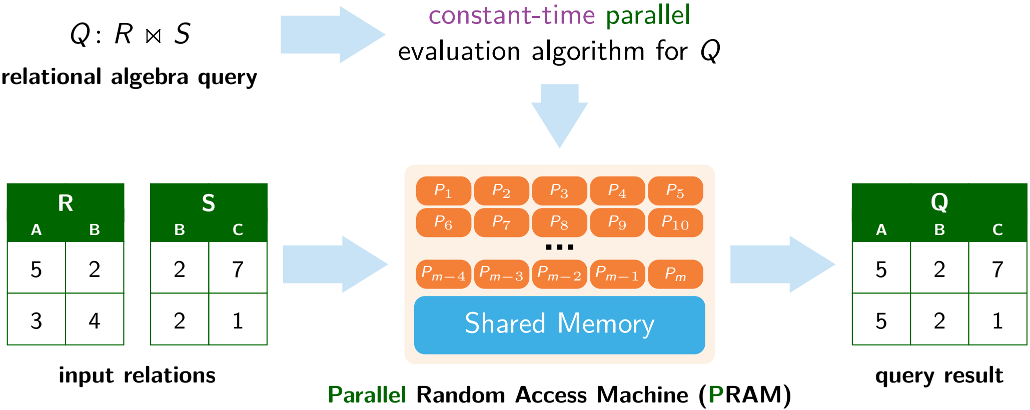
Random Access Machine (RAM)

Q		
A	B	C
5	2	7
5	2	1

query result

- **running time $\hat{=}$ number of computation steps**

Constant-Time Parallel Query Evaluation



$Q\colon R \bowtie S$

relational algebra query

constant-time parallel evaluation algorithm for Q

R	
A	**B**
5	2
3	4

S	
B	**C**
2	7
2	1

input relations

Parallel Random Access Machine (PRAM)

Q		
A	**B**	**C**
5	2	7
5	2	1

query result

Constant-Time Parallel Query Evaluation

$Q\colon R \bowtie S$

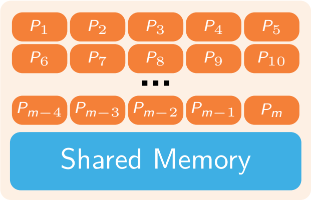
relational algebra query

constant-time parallel
evaluation algorithm for Q

R	
A	B
5	2
3	4

S	
B	C
2	7
2	1

input relations



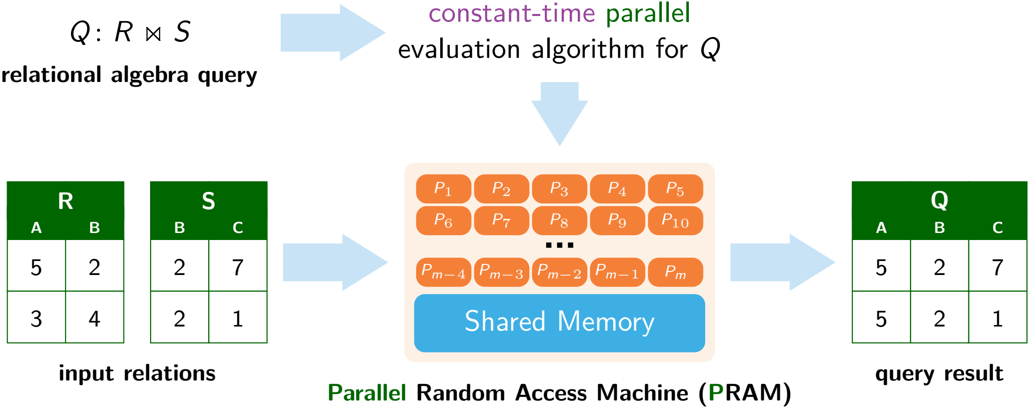
Parallel Random Access Machine (PRAM)

- **runs in constant time**
- **total number of computation steps?**

Q		
A	B	C
5	2	7
5	2	1

query result

Constant-Time Parallel Query Evaluation



$Q\colon R \bowtie S$

relational algebra query

constant-time parallel evaluation algorithm for Q

R	
A	B
5	2
3	4

S	
B	C
2	7
2	1

input relations

Parallel Random Access Machine (PRAM)

Q		
A	B	C
5	2	7
5	2	1

query result

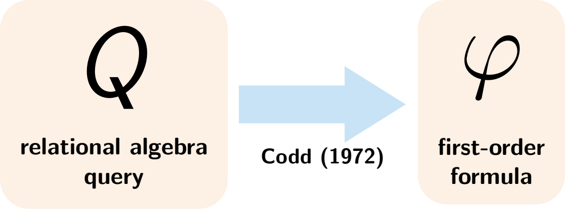
Work: Sum of computation steps of all processors

Constant-Time Parallel Evaluation

Q

relational algebra query

Constant-Time Parallel Evaluation



Q

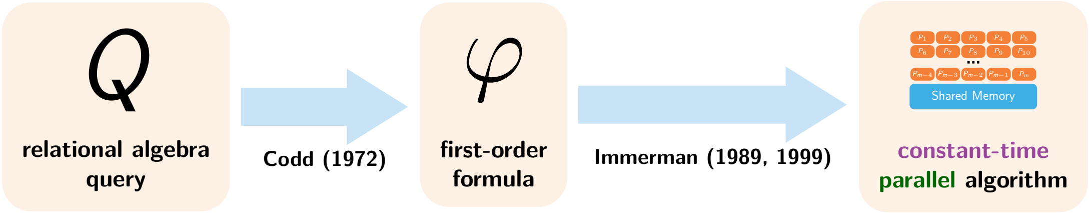
relational algebra query

Codd (1972)

φ

first-order formula

Constant-Time Parallel Evaluation



Q

relational algebra query

Codd (1972)

φ

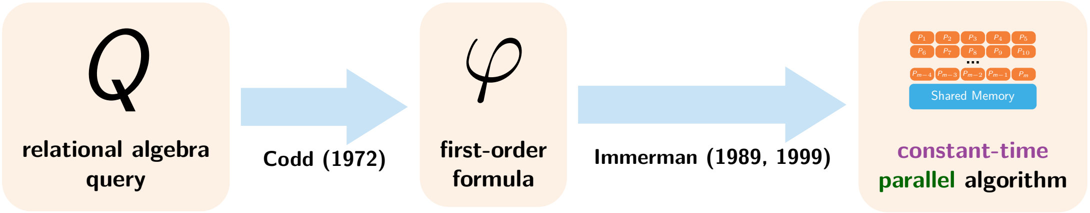
first-order formula

Immerman (1989, 1999)

constant-time parallel algorithm

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

Constant-Time Parallel Evaluation



Q

relational algebra query

Codd (1972)

φ

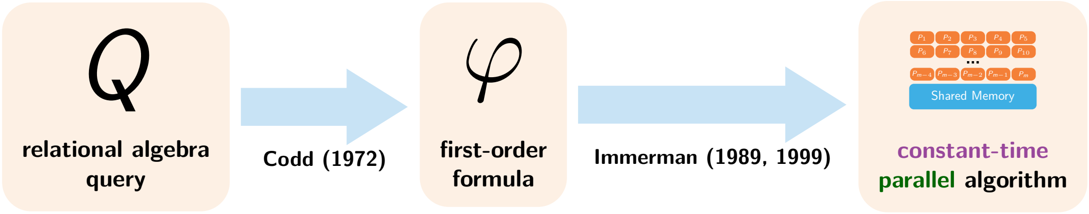
first-order formula

Immerman (1989, 1999)

constant-time parallel algorithm

- The resulting algorithm requires work $\mathcal{O}(n^k)$
  where k is the number of variables of the formula φ (Immerman 1989, 1999).

Constant-Time Parallel Evaluation


P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

Q

relational algebra query

Codd (1972)

φ

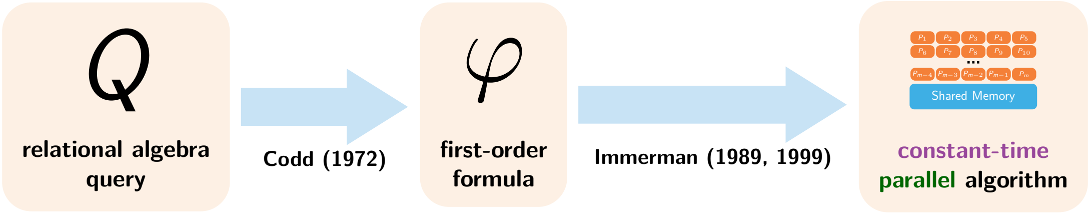
first-order formula

Immerman (1989, 1999)

constant-time parallel algorithm

- The resulting algorithm requires work $\mathcal{O}(n^k)$
  where k is the number of variables of the formula φ (Immerman 1989, 1999).
- Not work-optimal for many classes of queries

Constant-Time Parallel Evaluation



Q

relational algebra query

Codd (1972)

φ

first-order formula

Immerman (1989, 1999)

constant-time parallel algorithm

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

- The resulting algorithm requires work $\mathcal{O}(n^k)$ where k is the number of variables of the formula φ (Immerman 1989, 1999).
- Not work-optimal for many classes of queries

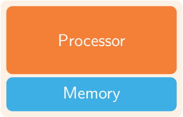
Definition

A constant time parallel algorithm is work-optimal if its work matches the running time of the best sequential algorithm.

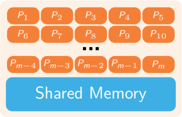
Overview of Main Results

Q

query class



classic, sequential RAM (known)

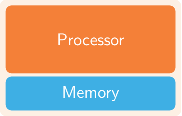


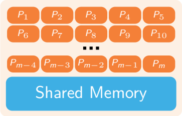
constant time, PRAM

assumptions/ data structures

Overview of Main Results

Q

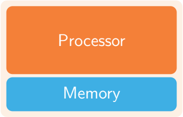


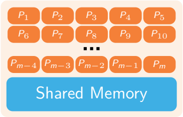

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

query class	classic, sequential RAM (known)	constant time, PRAM	assumptions/ data structures
semi-join algebra	time $\mathcal{O}(\mathsf{IN})$	work $\mathcal{O}(\mathsf{IN}^2)$	no assumptions

Overview of Main Results

$$Q$$

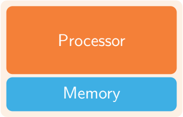


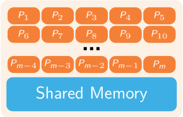

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

query class	**classic, sequential RAM (known)**	**constant time, PRAM**	**assumptions/ data structures**
semi-join algebra	**time $\mathcal{O}(\mathsf{IN})$**	**work $\mathcal{O}(\mathsf{IN})$**	**given a dictionary for database values**

Overview of Main Results

Q



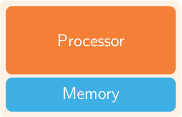

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

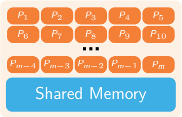
query class	classic, sequential RAM (known)	constant time, PRAM	assumptions/ data structures
semi-join algebra	**time $\mathcal{O}(\mathsf{IN})$**	**work $\mathcal{O}(\mathsf{IN})$**	**given a dictionary for database values**

work-optimal

Overview of Main Results

Q



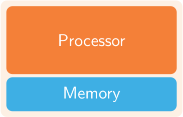

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

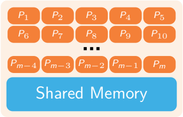
query class	classic, sequential RAM (known)	constant time, PRAM	assumptions/ data structures
semi-join algebra	time $\mathcal{O}(\textsf{IN})$	work $\mathcal{O}(\textsf{IN})$	given a dictionary for database values
acyclic conjunctive queries	time $\mathcal{O}(\textsf{IN} \cdot \textsf{OUT})$	work $\mathcal{O}((\textsf{IN} \cdot \textsf{OUT})^{1+\varepsilon})$	given a dictionary for database values

work-optimal

Overview of Main Results

Q

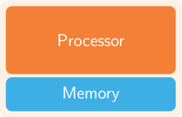


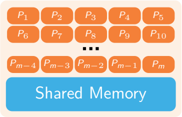

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

query class	**classic, sequential RAM (known)**	**constant time, PRAM**	**assumptions/ data structures**
semi-join algebra	**time** $\mathcal{O}(\textbf{IN})$	**work** $\mathcal{O}(\textbf{IN})$ **work-optimal**	**given a dictionary for database values**
acyclic conjunctive queries	**time** $\mathcal{O}(\textbf{IN} \cdot \textbf{OUT})$	**work** $\mathcal{O}((\textbf{IN} \cdot \textbf{OUT})^{1+\varepsilon})$ **work-efficient**	**given a dictionary for database values**

Overview of Main Results

Q

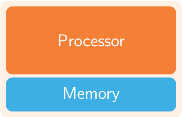


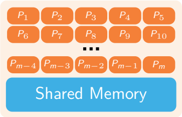

P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10}
...
P_{m-4} P_{m-3} P_{m-2} P_{m-1} P_m
Shared Memory

query class	classic, sequential RAM (known)	constant time, PRAM	assumptions/ data structures
semi-join algebra	time $\mathcal{O}(\mathsf{IN})$	work $\mathcal{O}(\mathsf{IN})$ work-optimal	given a dictionary for database values
acyclic conjunctive queries	time $\mathcal{O}(\mathsf{IN} \cdot \mathsf{OUT})$	work $\mathcal{O}((\mathsf{IN} \cdot \mathsf{OUT})^{1+\varepsilon})$ work-efficient	given a dictionary for database values
natural join queries (worst-case framework)	time $\mathcal{O}(\prod_{i=1}^{m} \lvert R_i \rvert^{x_i} + \mathsf{IN})$		

Overview of Main Results

Q





query class	classic, sequential RAM (known)	constant time, PRAM	assumptions/ data structures
semi-join algebra	time $\mathcal{O}(\mathsf{IN})$	work $\mathcal{O}(\mathsf{IN})$ work-optimal	given a dictionary for database values
acyclic conjunctive queries	time $\mathcal{O}(\mathsf{IN} \cdot \mathsf{OUT})$	work $\mathcal{O}((\mathsf{IN} \cdot \mathsf{OUT})^{1+\varepsilon})$ work-efficient	given a dictionary for database values
natural join queries (worst-case framework)	time $\mathcal{O}(\prod_{i=1}^{m} \lvert R_i \rvert^{x_i} + \mathsf{IN})$	work $\mathcal{O}((\prod_{i=1}^{m} \lvert R_i \rvert^{x_i} + \mathsf{IN})^{1+\varepsilon})$ work-efficient	given a dictionary for database values

Dictionaries

- dictionary-based compressed databases
- for instance, used for query optimisation (e.g. Chen, Gehrke, and Korn 2001)

Dictionaries

- dictionary-based compressed databases
- for instance, used for query optimisation (e.g. Chen, Gehrke, and Korn 2001)

Example

Movies	
Title	Year
True Romance	1993
Jurassic Park	1993
The Godfather	1972

Dictionaries

- dictionary-based compressed databases
- for instance, used for query optimisation (e.g. Chen, Gehrke, and Korn 2001)

Example

Movies	
Title	Year
True Romance	1993
Jurassic Park	1993
The Godfather	1972

→

Movies	
Title	Year
1	4
2	4
3	5

+

Dictionary	
Key	Value
1	True Romance
2	Jurassic Park
3	The Godfather
4	1993
5	1972

Dictionaries

- dictionary-based compressed databases
- for instance, used for query optimisation (e.g. Chen, Gehrke, and Korn 2001)

Example

Movies	
Title	Year
True Romance	1993
Jurassic Park	1993
The Godfather	1972

→

Movies	
Title	Year
1	4
2	4
3	5

+

Dictionary	
Key	Value
1	True Romance
2	Jurassic Park
3	The Godfather
4	1993
5	1972

- we require that all numbers are of size at most $\mathcal{O}(|D|)$ for a database D

Dictionaries

- dictionary-based compressed databases
- for instance, used for query optimisation (e.g. Chen, Gehrke, and Korn 2001)

Example

Movies	
Title	Year
True Romance	1993
Jurassic Park	1993
The Godfather	1972

Movies	
Title	Year
1	4
2	4
3	5

+

Dictionary	
Key	Value
1	True Romance
2	Jurassic Park
3	The Godfather
4	1993
5	1972

- we require that all numbers are of size at most $\mathcal{O}(|D|)$ for a database D

Settings for Constant-Time Parallel Evaluation

Dictionary Setting

- A dictionary for the database is available

Settings for Constant-Time Parallel Evaluation

Dictionary Setting

- A dictionary for the database is available

General Setting

- A single processor can test for equivalence in constant time
- Lemma: A dictionary can be computed in constant-time with work $\mathcal{O}(\mathsf{IN}^2)$.

Settings for Constant-Time Parallel Evaluation

Dictionary Setting

- A dictionary for the database is available

General Setting

- A single processor can test for equivalence in constant time
- Lemma: A dictionary can be computed in constant-time with work $\mathcal{O}(\mathsf{IN}^2)$.

Ordered Setting

- There is a linear order on the domain values
- A single processor can test for less than in constant time
- Lemma: For every $\varepsilon > 0$, a dictionary can be computed in constant-time with work $\mathcal{O}(\mathsf{IN}^{1+\varepsilon})$, given suitably ordered arrays for the database relations.

Settings

1. Work-Efficient Constant-Time Parallel Query Evaluation
   Preliminary results published at ICDT'23, Ioannina, Greece
   (Keppeler, Schwentick, and **S.** 2023)

data complexity

2. Parallel-Correctness and -Boundedness of Datalog Queries
   ICDT'19, Lisbon, Portugal (Neven, Schwentick, **S.**, and Vandevoort 2019)

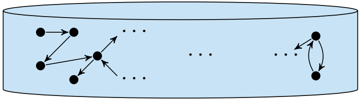
static analysis

3. Structurally Simple Rewritings

static analysis

Distributed Evaluation

global database



Query

- transitive closure T
- Datalog program

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation (fixed point computation)

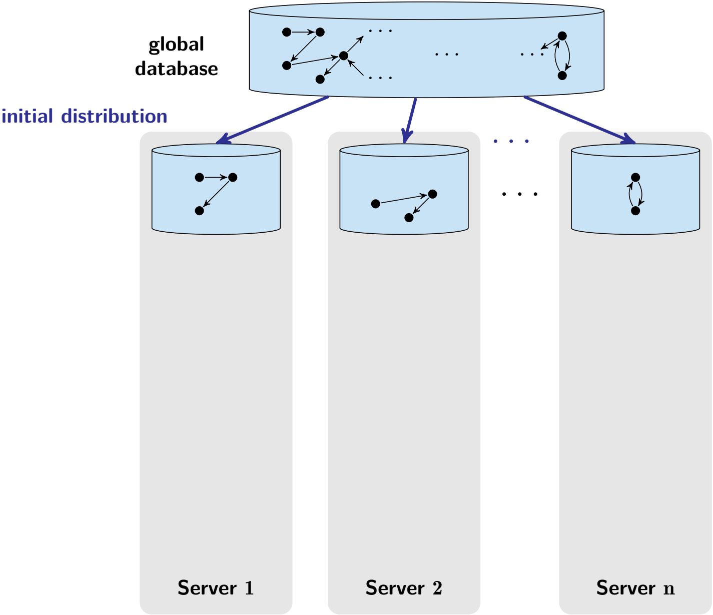
Distributed Evaluation

Query

- transitive closure T
- Datalog program

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation (fixed point computation)



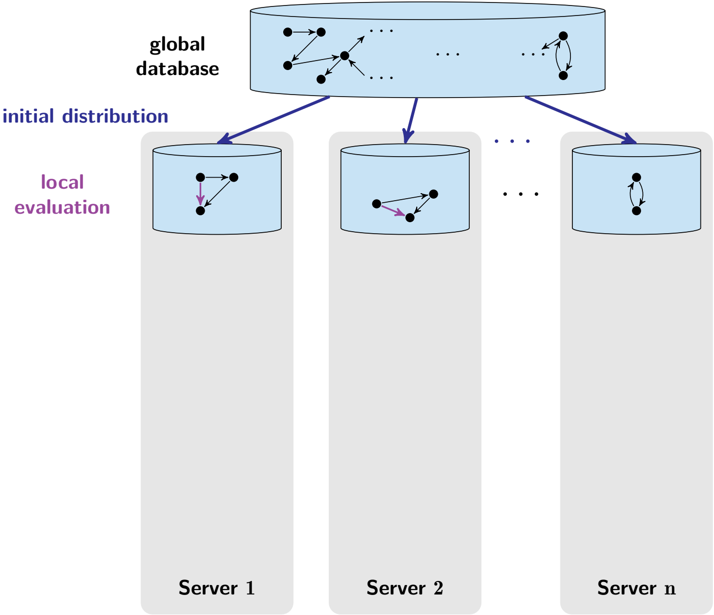
Distributed Evaluation

Query

- transitive closure T
- Datalog program

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation (fixed point computation)



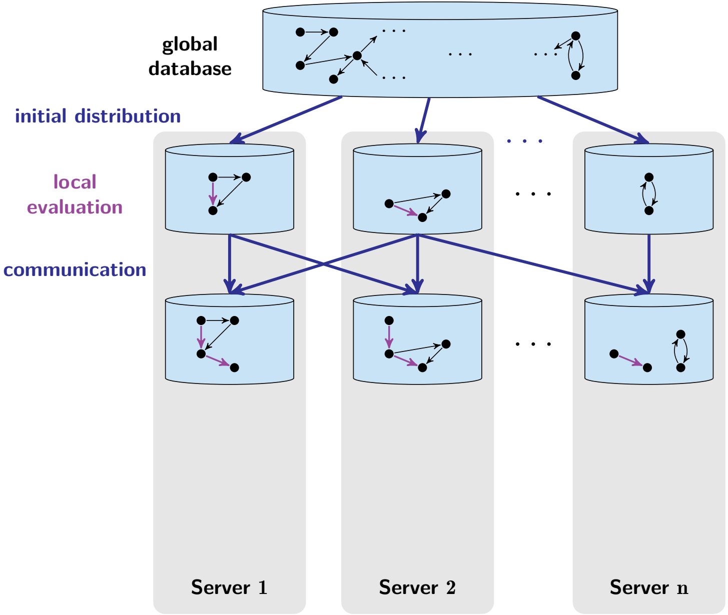
Distributed Evaluation

Query

- transitive closure T
- Datalog program

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation (fixed point computation)



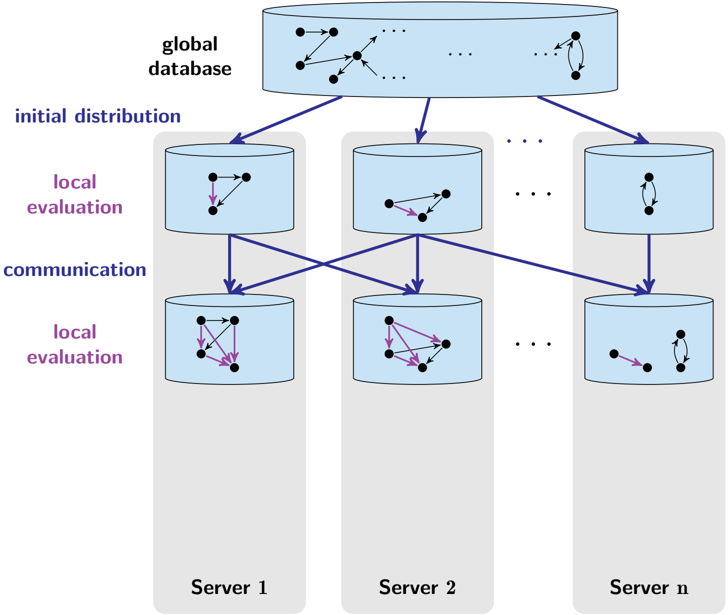
Distributed Evaluation

Query

- transitive closure T
- Datalog program

$$T(x, y) \leftarrow E(x, y)$$
$$T(x, z) \leftarrow T(x, y),\ E(y, z)$$

- recursive evaluation (fixed point computation)



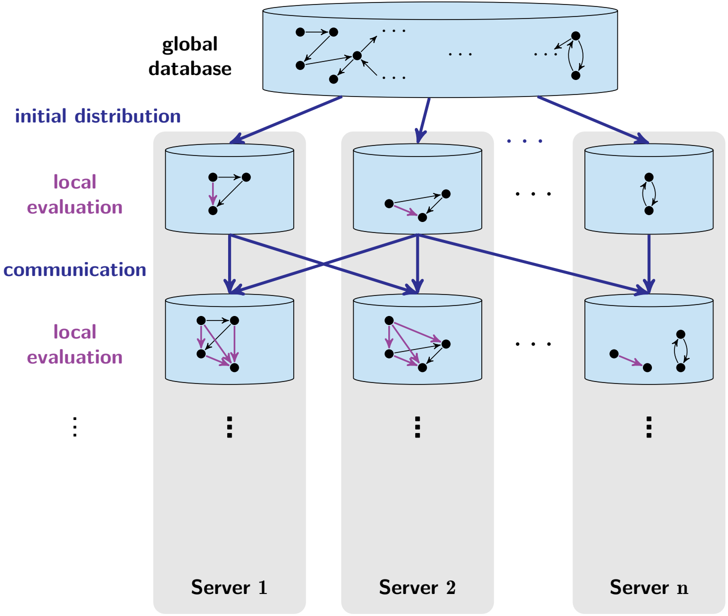
Distributed Evaluation

Query

- transitive closure T
- Datalog program

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation (fixed point computation)



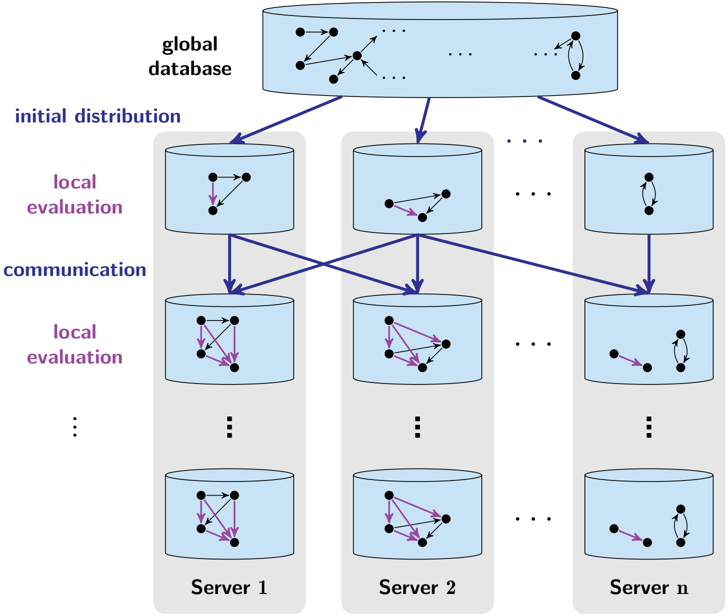
Distributed Evaluation

Query

- transitive closure T
- Datalog program

$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation (fixed point computation)



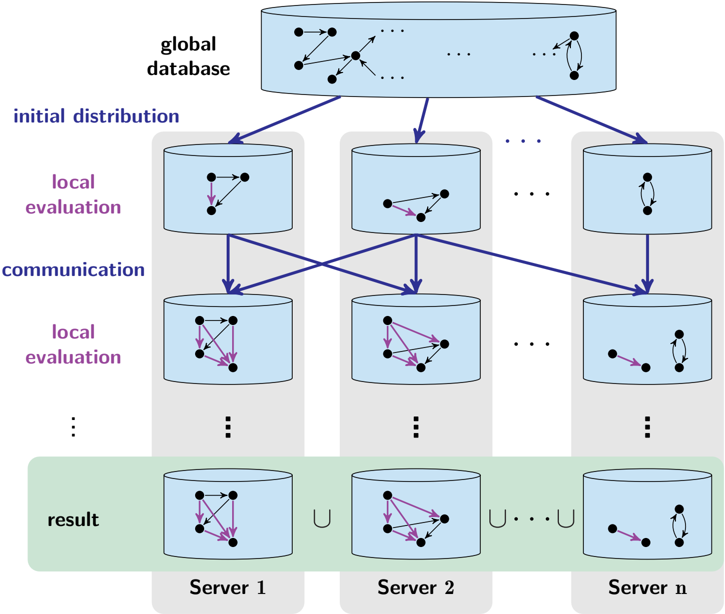
Distributed Evaluation

Query

- transitive closure T
- Datalog program

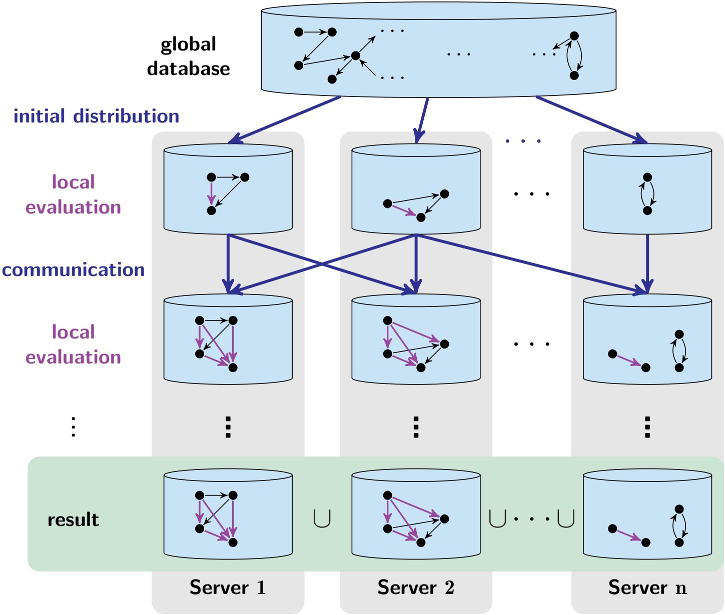
$$T(x,y) \leftarrow E(x,y)$$
$$T(x,z) \leftarrow T(x,y),\ E(y,z)$$

- recursive evaluation
  (fixed point computation)

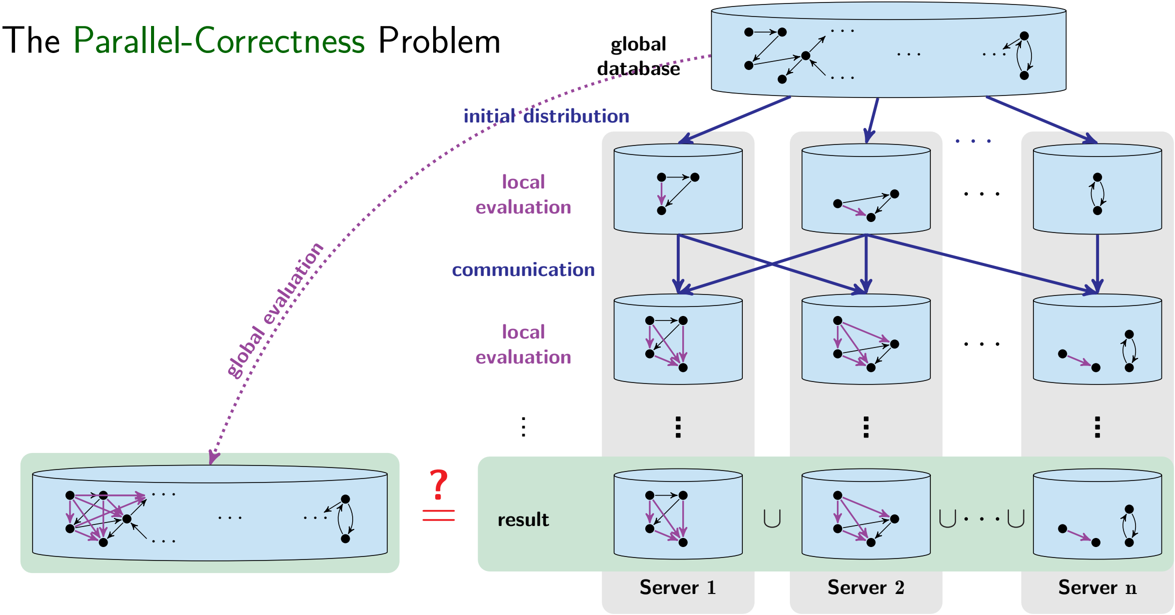


Distributed Evaluation

Massively Parallel Communication (MPC)
model
(Beame, Koutris, and Suciu 2017)



The Parallel-Correctness Problem



The Parallel-Correctness Problem

Parallel-Correctness Problem

Input:

- Datalog program
- distribution policy
- communication policy

Question:

Do distributed and global evaluation yield the same result for all databases?

The Parallel-Correctness Problem

Parallel-Correctness Problem

Input:

- Datalog program
- distribution policy
- communication policy

Question:

Do distributed and global evaluation yield the same result for all databases?

Theorem (Ketsman, Albarghouthi, and Koutris 2018)

Parallel-correctness for general Datalog programs is undecidable.

The Parallel-Correctness Problem

Parallel-Correctness Problem

Input:

- Datalog program
- distribution policy
- communication policy

Question:

Do distributed and global evaluation yield the same result for all databases?

Theorem (Ketsman, Albarghouthi, and Koutris 2018)

Parallel-correctness for general Datalog programs is undecidable.

- Even for “simple” policies:
  - only two servers
  - all but one relations are distributed to both servers
  - no communication

The Parallel-Correctness Problem

Parallel-Correctness Problem

Input:

- Datalog program
- distribution policy
- communication policy

Question:

Do distributed and global evaluation yield the same result for all databases?

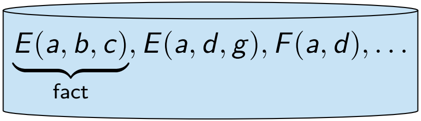
Theorem (Ketsman, Albarghouthi, and Koutris 2018)

Parallel-correctness for general Datalog programs is undecidable.

- Even for “simple” policies:
  - only two servers
  - all but one relations are distributed to both servers
  - no communication
- Is there a fragment of Datalog for which parallel-correctness is decidable?
- How to specify distribution and communication policies?

Basics

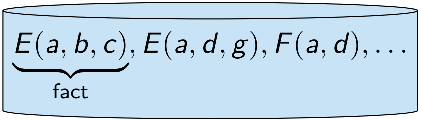
Relational databases



$$\underbrace{E(a,b,c)}_{\text{fact}}, E(a,d,g), F(a,d), \dots$$

Basics

Relational databases



$$\underbrace{E(a,b,c)}_{\text{fact}}, E(a,d,g), F(a,d), \ldots$$

Datalog programs consist of rules

$$\underbrace{T(x,y)}_{\text{head}} \leftarrow \underbrace{E(x,y,z), \overbrace{R(x,v)}^{\text{atom}}}_{\text{body}}.$$

- relation symbol of the head does not occur in the database
- rules can be recursive
- no negation

Parallel-Correctness and Containment

Undecidability of parallel-correctness results from the containment problem

... and containment is undecidable for general Datalog

Parallel-Correctness and Containment

Undecidability of parallel-correctness results from the containment problem

… and containment is undecidable for general Datalog

general Datalog

monadic Datalog

only unary head atoms

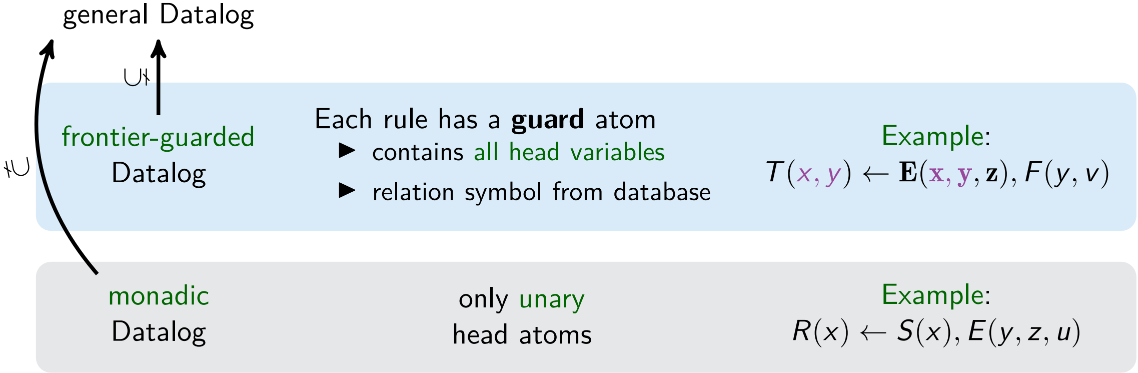
Example:
$R(x) \leftarrow S(x), E(y, z, u)$

Containment is decidable for monadic Datalog

Parallel-Correctness and Containment

Undecidability of parallel-correctness results from the containment problem

… and containment is undecidable for general Datalog



general Datalog

frontier-guarded Datalog

Each rule has a **guard** atom

- contains all head variables
- relation symbol from database

Example:
$T(x, y) \leftarrow \mathbf{E}(\mathbf{x}, \mathbf{y}, \mathbf{z}), F(y, v)$

monadic Datalog

only unary head atoms

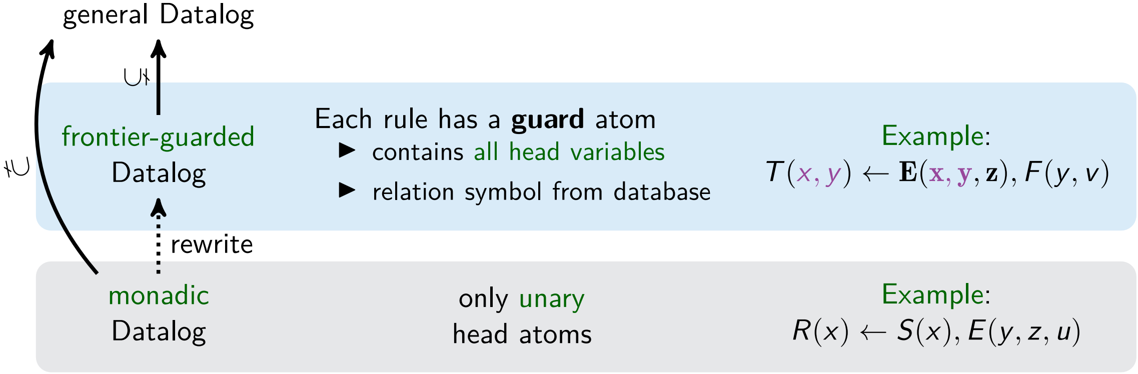
Example:
$R(x) \leftarrow S(x), E(y, z, u)$

Containment is decidable for monadic Datalog and frontier-guarded Datalog

Parallel-Correctness and Containment

Undecidability of parallel-correctness results from the containment problem

… and containment is undecidable for general Datalog



frontier-guarded Datalog

Each rule has a **guard** atom

- contains all head variables
- relation symbol from database

Example:

$T(x, y) \leftarrow \mathbf{E}(\mathbf{x}, \mathbf{y}, \mathbf{z}), F(y, v)$

monadic Datalog

only unary head atoms

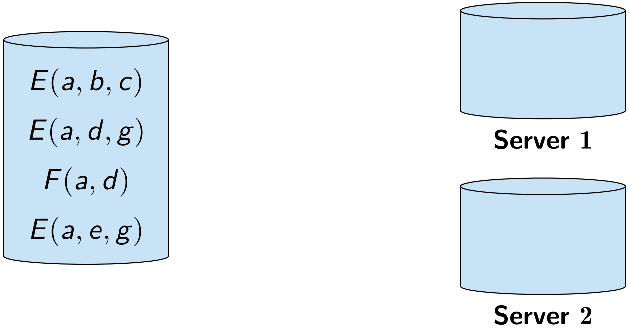
Example:

$R(x) \leftarrow S(x), E(y, z, u)$

Containment is decidable for monadic Datalog and frontier-guarded Datalog

Distribution Policies

Idea: Use hash functions $h_1, \ldots, h_k$ fast, evenly distribution



$E(a, b, c)$

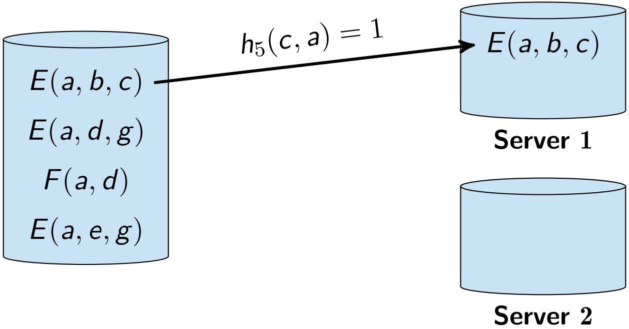
$E(a, d, g)$

$F(a, d)$

$E(a, e, g)$

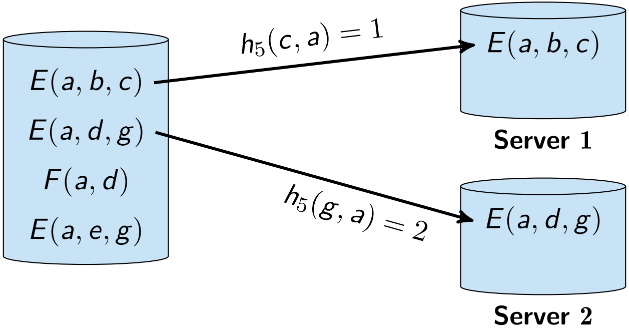
Distribution Policies

Idea: Use hash functions $h_1, \ldots, h_k$ fast, evenly distribution



Distribution Policies

Idea: Use hash functions $h_1, \ldots, h_k$ fast, evenly distribution



$E(a, b, c)$

$E(a, d, g)$

$F(a, d)$

$E(a, e, g)$

$h_5(c, a) = 1$

$h_5(g, a) = 2$

$E(a, b, c)$

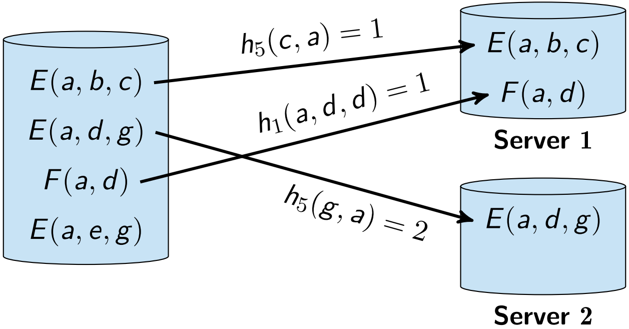
Server 1

$E(a, d, g)$

Server 2

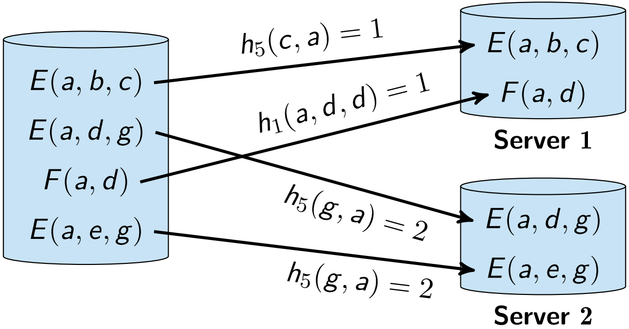
Distribution Policies

Idea: Use hash functions $h_1, \ldots, h_k$ fast, evenly distribution



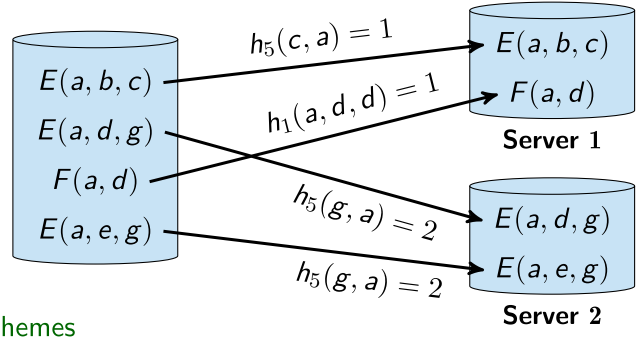
Distribution Policies

Idea: Use hash functions $h_1, \ldots, h_k$ fast, evenly distribution



Distribution Policies

Idea: Use hash functions $h_1, \dots, h_k$ fast, evenly distribution



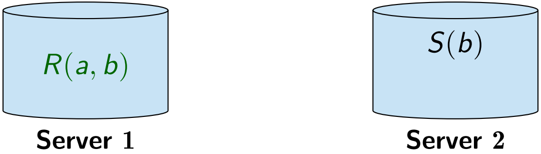
Here: Hash policy schemes

- describes how hash functions are applied
- defines class of hash functions

Communication Policies

Data-Moving Distribution Constraints

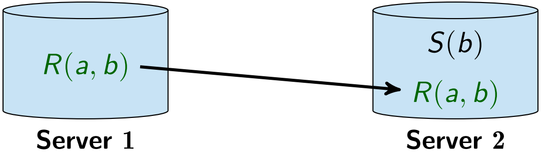
$$\underbrace{R(x,y)@\lambda, S(y)@\kappa}_{\text{body}} \rightarrow \underbrace{R(x,y)@\kappa}_{\text{head}}$$



Communication Policies

Data-Moving Distribution Constraints

$$\underbrace{R(x,y)@\lambda, S(y)@\kappa}_{\text{body}} \rightarrow \underbrace{R(x,y)@\kappa}_{\text{head}}$$



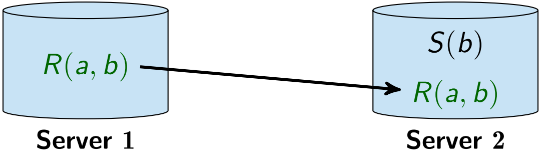
Communication Policies

Data-Moving Distribution Constraints

$$\underbrace{R(x,y)@\lambda, S(y)@\kappa}_{\text{body}} \rightarrow \underbrace{R(x,y)@\kappa}_{\text{head}}$$

Both $R(x,y)$ and κ occur in the body.

- No creation of facts
- No creation of servers



Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and data-moving distribution constraints
frontier-guarded	
monadic	

Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and data-moving distribution constraints
frontier-guarded	undecidable*
monadic	undecidable*

*mainly contributed by my co-authors to the ICDT'19 paper

Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and	...with polynomial communication property	
	data-moving distribution constraints	syntactical fragment	changed semantics
frontier-guarded	undecidable*		
monadic	undecidable*		

Polynomial Communication Property

- The amount of communication without any local computation in between is bounded polynomially

*mainly contributed by my co-authors to the ICDT'19 paper

Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and	…with polynomial communication property	
	data-moving distribution constraints	syntactical fragment	changed semantics
frontier-guarded	undecidable*	2ExpTime-complete	2ExpTime-complete
monadic	undecidable*		

Theorem

Parallel-correctness for frontier-guarded Datalog,

- *hash policy schemes, and*
- *data-moving distribution constraints*
- *with the polynomial communication property*

is 2ExpTime-complete.

*mainly contributed by my co-authors to the ICDT'19 paper

Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and data-moving distribution constraints	…with polynomial communication property	
		syntactical fragment	changed semantics
frontier-guarded	undecidable*	2ExpTime-complete	2ExpTime-complete
monadic	undecidable*	in 2ExpTime	in 2ExpTime

Reminder: Every monadic Datalog query can be translated into an equivalent frontier-guarded Datalog query.

*mainly contributed by my co-authors to the ICDT'19 paper

Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and	…with polynomial communication property	
	data-moving distribution constraints	syntactical fragment	changed semantics
frontier-guarded	undecidable[*]	2ExpTime-complete	2ExpTime-complete
monadic	undecidable[*]	~~in 2ExpTime~~	~~in 2ExpTime~~

Reminder: Every monadic Datalog query can be translated into an equivalent frontier-guarded Datalog query.

[*] mainly contributed by my co-authors to the ICDT'19 paper

Parallel-Correctness: Main Results

Datalog fragment	hash policy schemes and data-moving distribution constraints	…with polynomial communication property	
		syntactical fragment	changed semantics
frontier-guarded	undecidable*	2ExpTime-complete	2ExpTime-complete
monadic	undecidable*	open	undecidable*

Reminder: Every monadic Datalog query can be translated into an equivalent frontier-guarded Datalog query.

*mainly contributed by my co-authors to the ICDT'19 paper

Parallel-Boundedness

Parallel-Boundedness

There is a bound $r \in \mathbb{N}$ such that

- for every database
- no new facts are computed
- after r communication rounds.

- Local computations may be unbounded!

Parallel-Boundedness

Parallel-Boundedness

There is a bound $r \in \mathbb{N}$ such that

- for every database
- no new facts are computed
- after r communication rounds.

- Local computations may be unbounded!

Theorem

Parallel-boundedness for frontier-guarded Datalog programs,

- *hash policy schemes, and*
- *data-moving distribution constraints with the polynomial communication property*

that are parallel-correct is 2ExpTime-*complete.*

Settings

1. Work-Efficient Constant-Time Parallel Query Evaluation
   Preliminary results published at ICDT'23, Ioannina, Greece
   (Keppeler, Schwentick, and **S.** 2023)

data complexity

---

2. Parallel-Correctness and -Boundedness of Datalog Queries
   ICDT'19, Lisbon, Portugal (Neven, Schwentick, **S.**, and Vandevoort 2019)

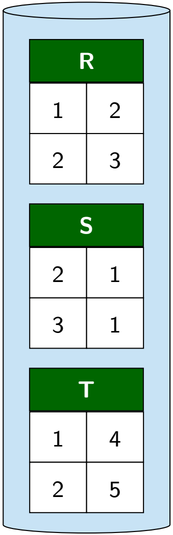
static analysis

---

3. Structurally Simple Rewritings
   ICDT'22, Edinburgh, UK (Geck, Keppeler, Schwentick, and **S.** 2022)
   LMCS Journal (Geck, Keppeler, Schwentick, and **S.** 2023)

static analysis

Rewritings



R	
1	2
2	3

S	
2	1
3	1

T	
1	4
2	5

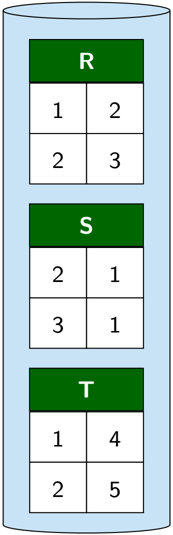
relational database

Rewritings

Query $H(x, w) \leftarrow R(x, y),\ S(y, z),\ T(z, w)$

Conjunctive Query

single, non-recursive rule



R	
1	2
2	3

S	
2	1
3	1

T	
1	4
2	5

relational database

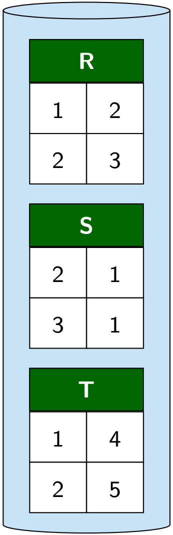
Rewritings

no direct access

Query $H(x, w) \leftarrow R(x, y),\ S(y, z),\ T(z, w)$

Conjunctive Query

single, non-recursive rule



R	
1	2
2	3

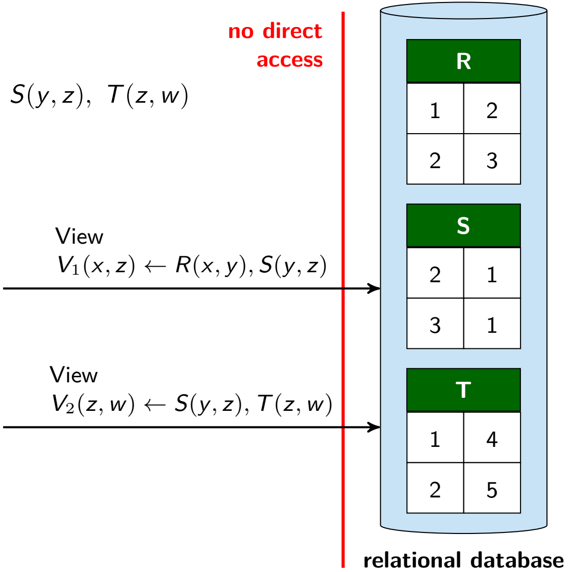
S	
2	1
3	1

T	
1	4
2	5

relational database

Rewritings

Query $H(x, w) \leftarrow R(x, y),\ S(y, z),\ T(z, w)$



no direct access

View
$V_1(x, z) \leftarrow R(x, y), S(y, z)$

View
$V_2(z, w) \leftarrow S(y, z), T(z, w)$

R	
1	2
2	3

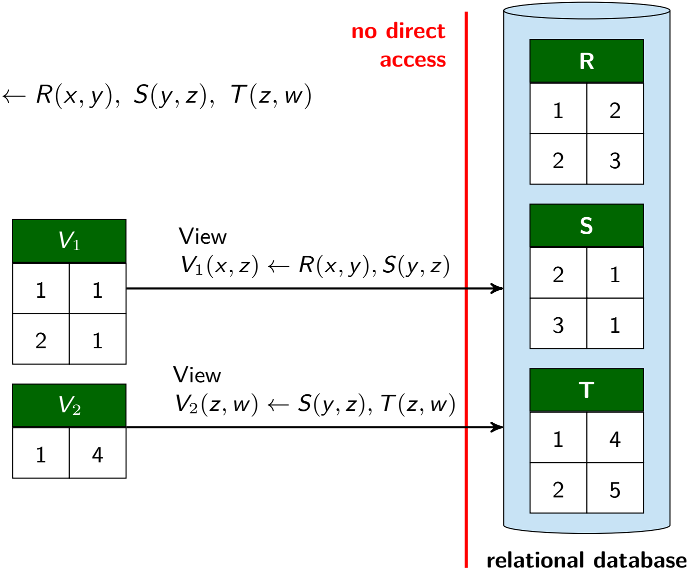
S	
2	1
3	1

T	
1	4
2	5

relational database

Rewritings

Query $H(x, w) \leftarrow R(x, y),\ S(y, z),\ T(z, w)$



View
$V_1(x, z) \leftarrow R(x, y), S(y, z)$

View
$V_2(z, w) \leftarrow S(y, z), T(z, w)$

V_1	
1	1
2	1

V_2	
1	4

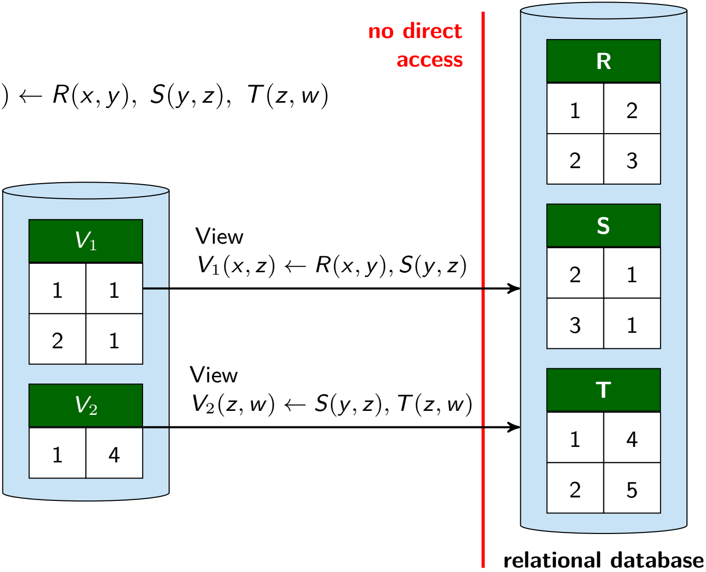
R	
1	2
2	3

S	
2	1
3	1

T	
1	4
2	5

Rewritings

Query $H(x, w) \leftarrow R(x, y),\ S(y, z),\ T(z, w)$



no direct access

View $V_1(x, z) \leftarrow R(x, y), S(y, z)$

View $V_2(z, w) \leftarrow S(y, z), T(z, w)$

V_1	
1	1
2	1

V_2	
1	4

R	
1	2
2	3

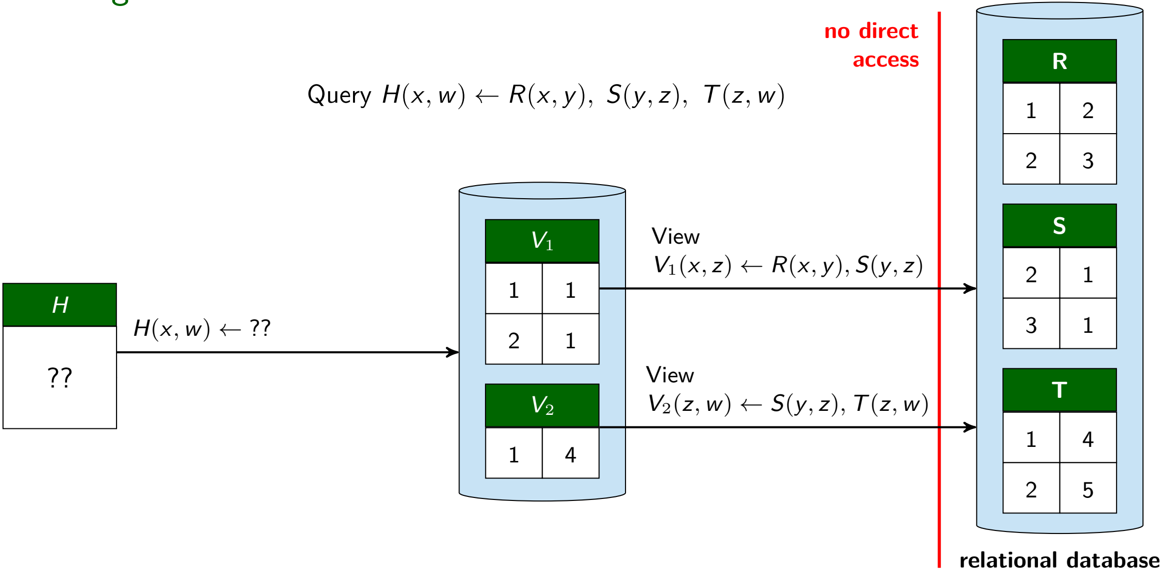
S	
2	1
3	1

T	
1	4
2	5

relational database

Rewritings

Query $H(x,w) \leftarrow R(x,y),\ S(y,z),\ T(z,w)$



H
??

$H(x,w) \leftarrow ??$

V_1	
1	1
2	1

V_2	
1	4

View
$V_1(x,z) \leftarrow R(x,y), S(y,z)$

View
$V_2(z,w) \leftarrow S(y,z), T(z,w)$

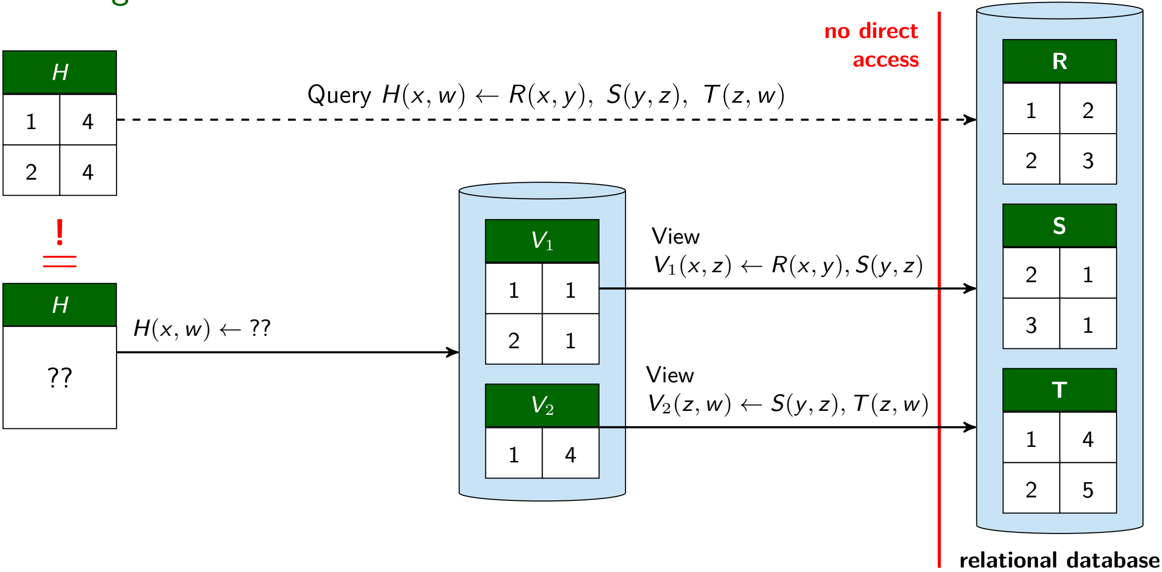
R	
1	2
2	3

S	
2	1
3	1

T	
1	4
2	5

relational database

Rewritings



H	
1	4
2	4

Query $H(x,w) \leftarrow R(x,y),\ S(y,z),\ T(z,w)$

H
??

$H(x,w) \leftarrow ??$

V_1	
1	1
2	1

V_2	
1	4

View $V_1(x,z) \leftarrow R(x,y), S(y,z)$

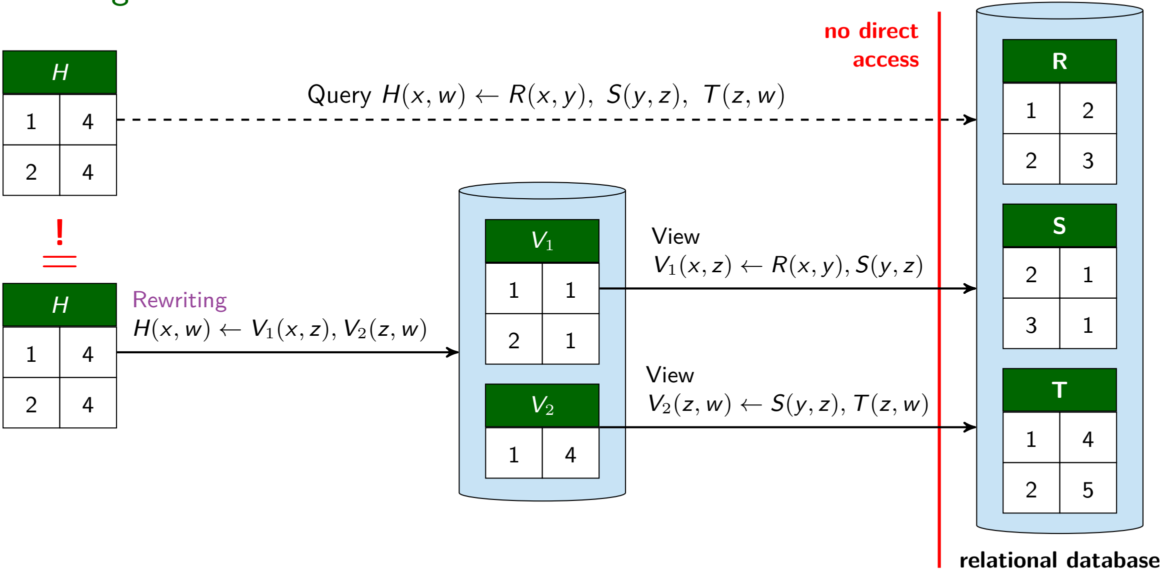
View $V_2(z,w) \leftarrow S(y,z), T(z,w)$

R	
1	2
2	3

S	
2	1
3	1

T	
1	4
2	5

Rewritings



H	
1	4
2	4

Query $H(x,w) \leftarrow R(x,y),\ S(y,z),\ T(z,w)$

no direct access

$\overset{!}{=}$

H	
1	4
2	4

Rewriting
$H(x,w) \leftarrow V_1(x,z), V_2(z,w)$

V_1	
1	1
2	1

V_2	
1	4

View
$V_1(x,z) \leftarrow R(x,y), S(y,z)$

View
$V_2(z,w) \leftarrow S(y,z), T(z,w)$

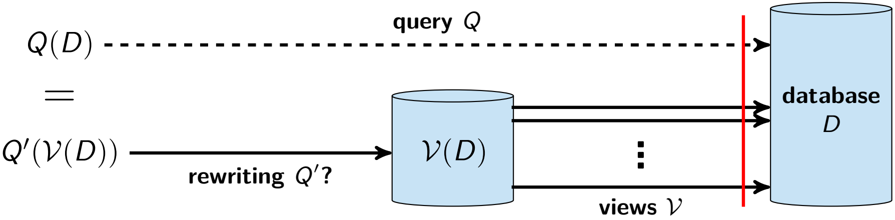
R	
1	2
2	3

S	
2	1
3	1

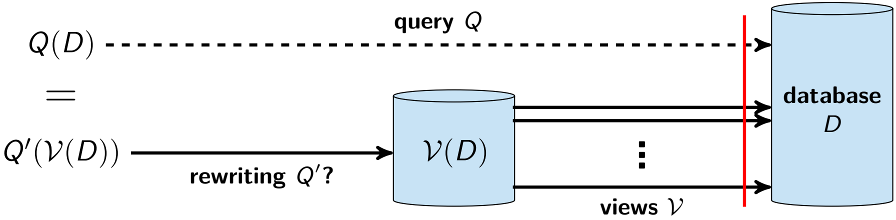
T	
1	4
2	5

relational database

The Rewriting Problem



The Rewriting Problem



The Rewriting Problem

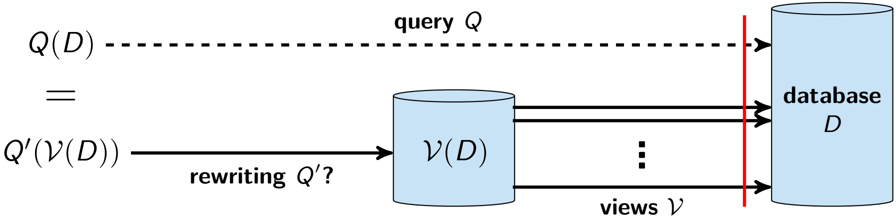
Input:

- conjunctive query Q
- set $\mathcal{V}$ of views

Question:

Is there a rewriting for Q with respect to $\mathcal{V}$?

The Rewriting Problem



The Rewriting Problem

Input:

- conjunctive query Q
- set $\mathcal{V}$ of views

Question:

Is there a rewriting for Q with respect to $\mathcal{V}$?

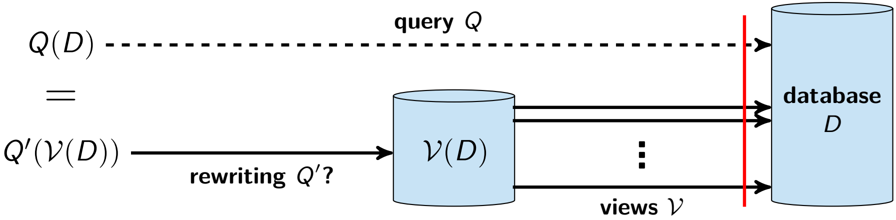
Theorem (Levy et al. 1995)

The rewriting problem for

- *conjunctive queries and*
- *views defined by conjunctive queries*

is NP-complete.

The Rewriting Problem



The Rewriting Problem

Input:

- conjunctive query Q
- set $\mathcal{V}$ of views

Question:

Is there a rewriting for Q with respect to $\mathcal{V}$?

Theorem (Levy et al. 1995)

The rewriting problem for

- *conjunctive queries and*
- *views defined by conjunctive queries*

is NP-complete.

→ Restrict everything to structurally simple queries

Acyclic Conjunctive Queries

For acyclic queries many problems are in polynomial time: containment, evaluation, ...

Acyclic Conjunctive Queries

For acyclic queries many problems are in polynomial time: containment, evaluation, ...

Definition

A conjunctive query is acyclic if it has a join tree

Acyclic Conjunctive Queries

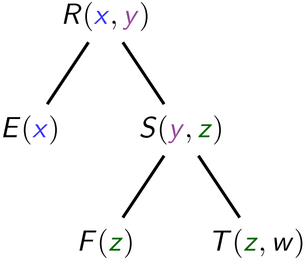
For acyclic queries many problems are in polynomial time: containment, evaluation, ...

Definition

A conjunctive query is acyclic if it has a join tree

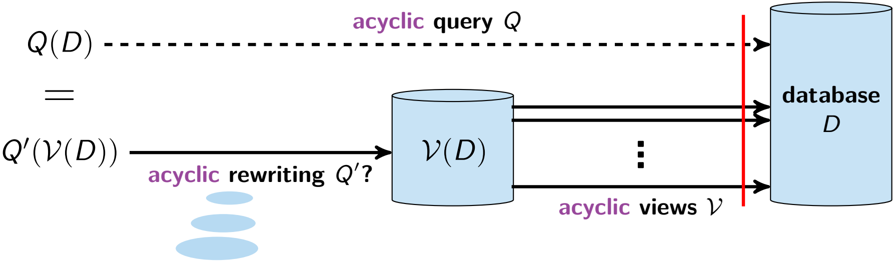
Example

$H(x, y) \leftarrow R(x, y), S(y, z), F(z), E(x), T(z, w)$ is acyclic



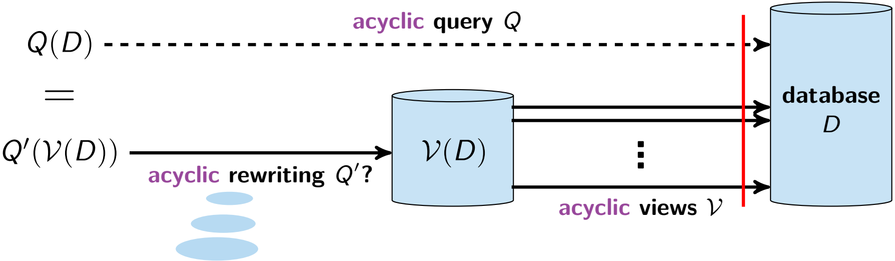
For every variable:
the induced subgraph is connected

Complexity of the Acyclic Rewriting Problem



If the query is acyclic, we would like the rewriting to be acyclic as well

Complexity of the Acyclic Rewriting Problem



If the query is acyclic, we would like the rewriting to be acyclic as well

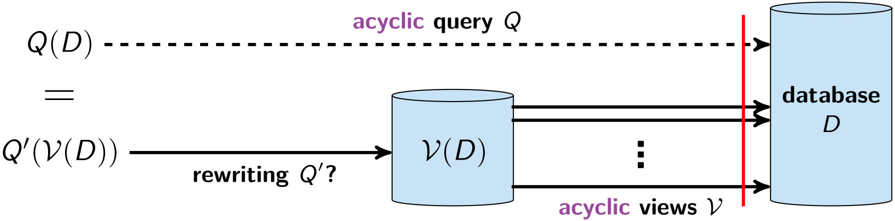
Theorem

The acyclic rewriting problem for

- ▶ *acyclic queries and*
- ▶ *views defined by acyclic queries*

is NP-complete.

Complexity of the Acyclic Rewriting Problem



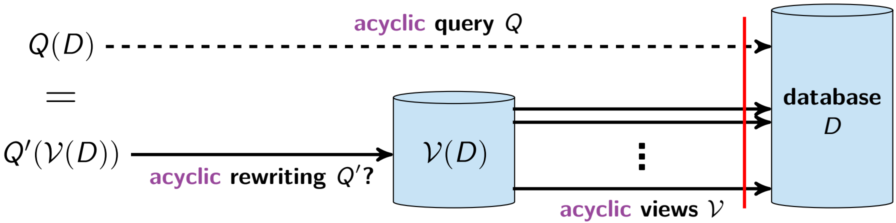
Theorem

The rewriting problem for

- *acyclic queries and*
- *views defined by acyclic queries*

is NP-complete.

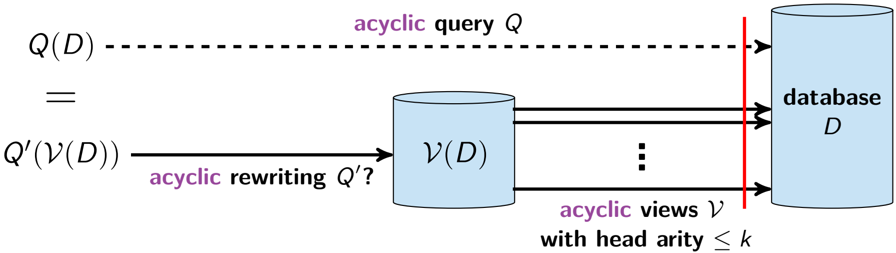
Complexity of the Acyclic Rewriting Problem



Theorem

If the query is acyclic and there is any rewriting, there is an acyclic rewriting.

Complexity of the Acyclic Rewriting Problem



Theorem

For every $k \geq 0$, the acyclic rewriting problem for

- *acyclic queries and*
- *views defined by acyclic queries with head arity at most k*

is in polynomial time.

Rewritings: Main Results

Views	Query	Rewriting	Restriction of views	arity of database relations	
				is $\le k, k \in \mathbb{N}_0$	unbounded
acyclic	acyclic	acyclic	no restriction	NP-complete for $k \geq 3$	
acyclic	acyclic	acyclic	head arity $\leq \ell$ $\ell \in \mathbb{N}_0$	polynomial time	
acyclic	acyclic	acyclic	weak head arity $\leq \ell$ $\ell \in \mathbb{N}_0$	polynomial time	
free-connex acyclic	acyclic	acyclic	no restriction	polynomial time	open
hierarchical	hierarchical	hierarchical	no restriction	NP-complete for $k \geq 3$	
q-hierarchical	q-hierarchical	q-hierarchical	no restriction	polynomial time	open

Conclusion

1. Work-Efficient Constant-Time Parallel Query Evaluation
   - Transforming classical algorithms into constant-time parallel algorithms

data complexity

2. Parallel-Correctness and -Boundedness of Datalog Queries
   - Deciding whether query evaluation is correct

static analysis

3. Structurally Simple Rewritings
   - Preserving structural properties of queries under access restriction

static analysis

References

Beame, Paul, Paraschos Koutris, and Dan Suciu (2017). “Communication Steps for Parallel Query Processing.” In: *Journal of the ACM* 64.6, 40:1–40:58. doi: 10.1145/3125644.

Chen, Zhiyuan, Johannes Gehrke, and Flip Korn (2001). “Query Optimization In Compressed Database Systems.” In: *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*. Ed. by Sharad Mehrotra and Timos K. Sellis. ACM, pp. 271–282. doi: 10.1145/375663.375692.

Codd, E. F. (1972). “Relational Completeness of Data Base Sublanguages.” In: *Database Systems*. Ed. by R. Rustin. Prentice-Hall, pp. 33–64.

Geck, Gaetano, Jens Keppeler, Thomas Schwentick, and Christopher **S.** (2022). “Rewriting with Acyclic Queries: Mind Your Head.” In: *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*. Ed. by Dan Olteanu and Nils Vortmeier. Vol. 220. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:20. doi: 10.4230/LIPIcs.ICDT.2022.8. url: https://doi.org/10.4230/LIPIcs.ICDT.2022.8.

References (cont.)

- Geck, Gaetano, Jens Keppeler, Thomas Schwentick, and Christopher **S.** (2023). "Rewriting with Acyclic Queries: Mind Your Head." In: *Logical Methods in Computer Science* 19.4. doi: 10.46298/LMCS-19(4:17)2023. url: https://doi.org/10.46298/lmcs-19(4:17)2023.
- Immerman, Neil (1989). "Expressibility and Parallel Complexity." In: *SIAM Journal on Computing* 18.3, pp. 625–638. doi: 10.1137/0218043.
- Immerman, Neil (1999). *Descriptive Complexity*. Graduate texts in computer science. Springer. doi: 10.1007/978-1-4612-0539-5.
- Keppeler, Jens, Thomas Schwentick, and Christopher **S.** (2023). "Work-Efficient Query Evaluation with PRAMs." In: *26th International Conference on Database Theory, ICDT 2023, March 28-31, 2023, Ioannina, Greece*. Ed. by Floris Geerts and Brecht Vandevoort. Vol. 255. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 16:1–16:20. doi: 10.4230/LIPIcs.ICDT.2023.16. url: https://doi.org/10.4230/LIPIcs.ICDT.2023.16.
- Ketsman, Bas, Aws Albarghouthi, and Paraschos Koutris (2018). "Distribution Policies for Datalog." en. In: *International Conference on Database Theory, ICDT 2018*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 17:1–17:22. doi: 10.4230/LIPIcs.ICDT.2018.17.

References (cont.)

- Levy, Alon Y., Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava (1995). “Answering Queries Using Views.” In: *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*. Ed. by Mihalis Yannakakis and Serge Abiteboul. ACM Press, pp. 95–104. doi: 10.1145/212433.220198.
- Neven, Frank, Thomas Schwentick, Christopher **S.**, and Brecht Vandevoort (2019). “Parallel-Correctness and Parallel-Boundedness for Datalog Programs.” In: *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*. Ed. by Pablo Barceló and Marco Calautti. Vol. 127. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:19. doi: 10.4230/LIPIcs.ICDT.2019.14. url: https://doi.org/10.4230/LIPIcs.ICDT.2019.14.