ICDE 2022 tutorial



Toward Responsive DBMS: Optimal Join Algorithms, Enumeration, Factorization, Ranking, and Dynamic Programming

Nikolaos Tziavelis, Wolfgang Gatterbauer, Mirek Riedewald

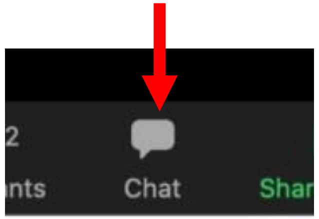
Northeastern University, Boston

Part 1 : Introduction

Slides: https://northeastern-datalab.github.io/responsive-dbms-tutorial
DOI: https://doi.org/10.1109/ICDE53745.2022.00299
Data Lab: https://db.khoury.northeastern.edu

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 4.0 International License.
See https://creativecommons.org/licenses/by-nc-sa/4.0/ for details

Recommended Zoom interaction (also posted in Slack)



- **Q&A**: Please use the chat for questions.
  One of two not presenting will answer in the box.
- **Post-tutorial Zoom space:** We have 30min after the tutorial for discussion on the same zoom space
- We love feedback. If you have questions / comments / concerns after the tutorial today, please contact us. We are also happy to meet again later if there is interest.

Outline tutorial

1: Introduction (Nikos) ~40min

2: Tree Decompositions (Mirek) ~20min

3: Acyclic Queries & Enumeration (Wolfgang) ~25min

BREAK

4: Factorization (Nikos) ~10min

5: Dynamic Programming & Semirings (Wolfgang) ~20min

6: Any-k or Ranked Enumeration (Nikos) ~35min

7. Decomposition of Comparison Predicates (Mirek) ~10min

8. Conclusion (Mirek) ~10min

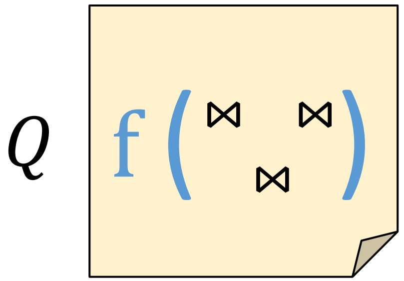
Outline Part 1

Part 1: Introduction

- What is this tutorial about?
- Overview of Queries/Tasks
- Measures of Success
- Overview of Techniques

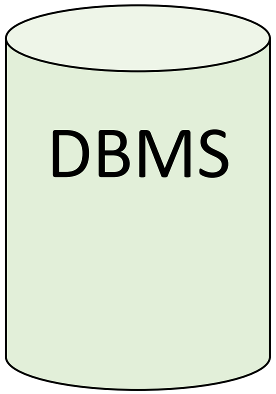
Why "Responsive DBMS"

- Database systems can become unresponsive when submitting a query, not returning anything in the output for a long time
- This is often the case when the query involves the join of many tables
- However, the user might not be interested in the entire join output, but in some task f() over the join



Example tasks f():
- Count #join answers
- Find top-k answers
- Find median answer
- …

Typical DBMS strategy:
first materialize the join,
then perform f()

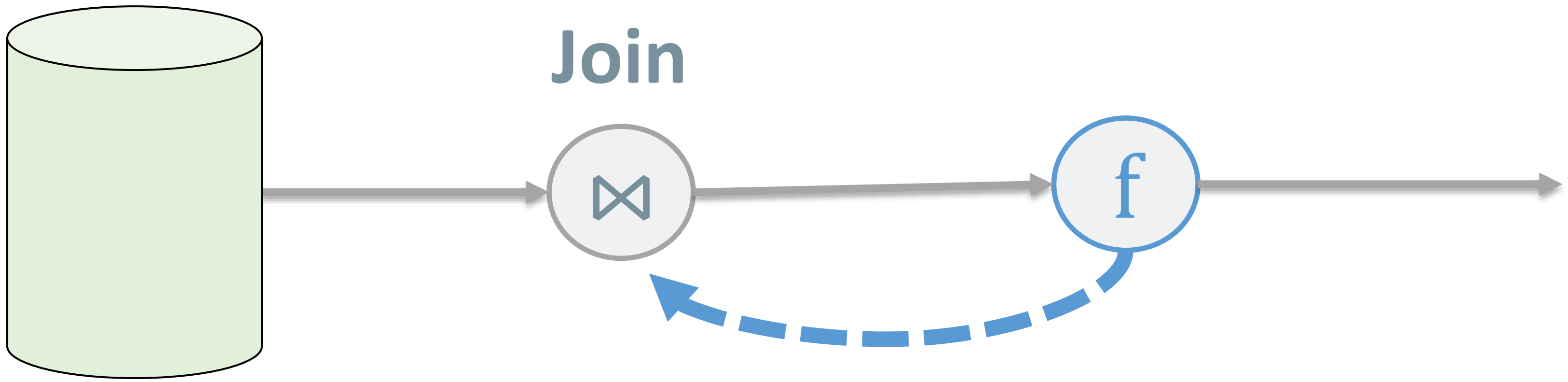


…

Why "Responsive DBMS"

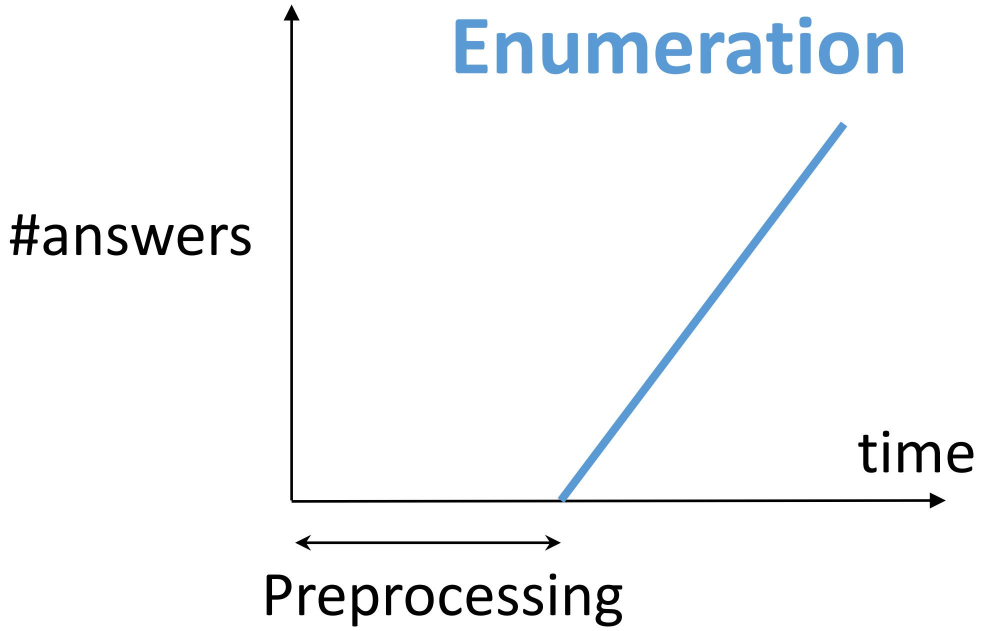
- How do we cope with this problem?
- Strategy 1: push f() into the join

Can we perform the task f() **without** (and significantly faster than) materializing the join?



Why "Responsive DBMS"

- How do we cope with this problem?
- Strategy 1: push f() into the join
- Strategy 2: if the output of f() is large, then enumerate



Can we return some answers back to the user before all the answers are available?

Relevant Concepts

Any-k

Enumeration

Dynamic Programming

Semirings

(Hyper)tree Decompositions

Aggregates

Selection/ Direct Access

Free-connex CQs

Factorization

Batch

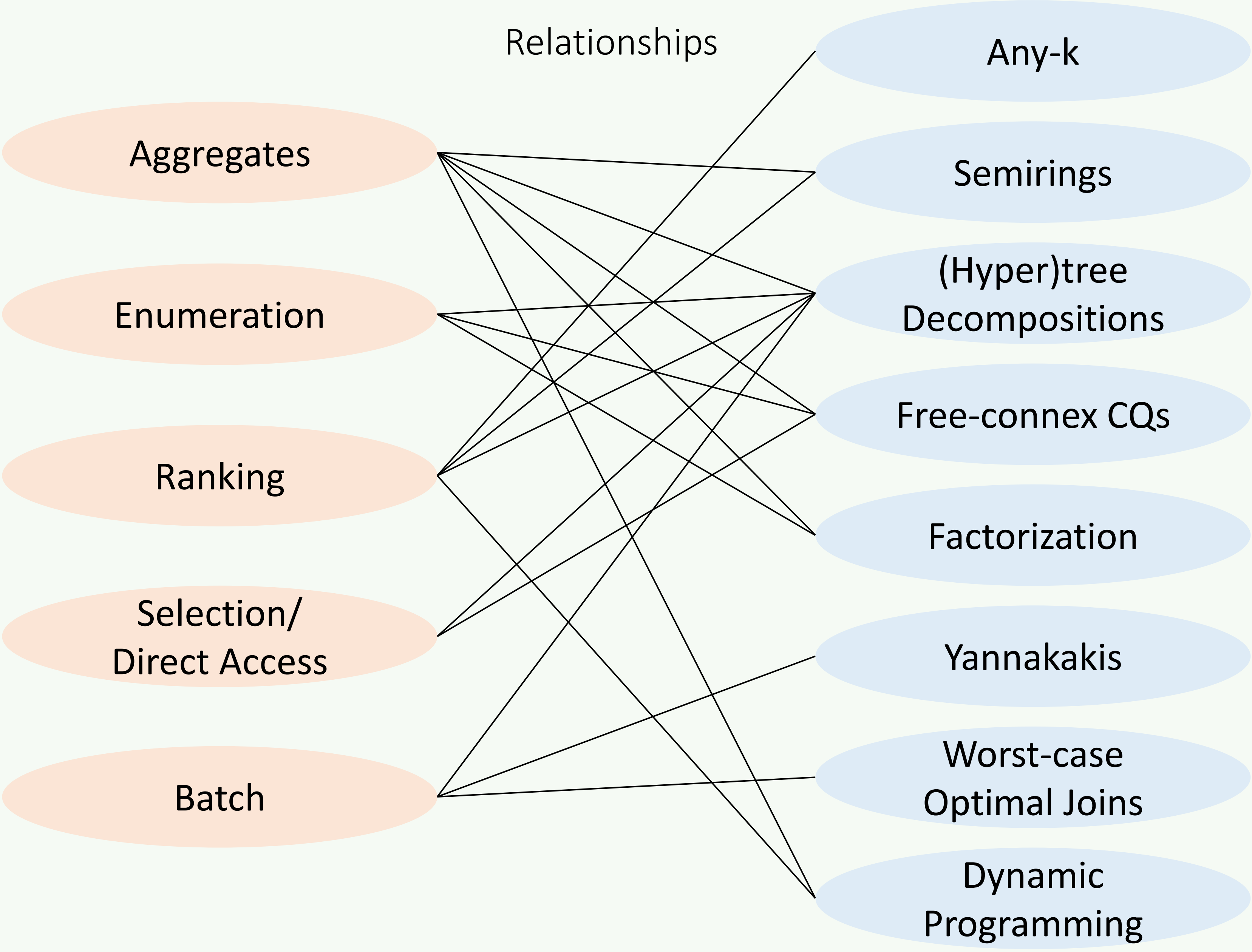
Yannakakis

Worst-case Optimal Joins

Ranking

Queries/Tasks

Techniques



Outline Part 1

Part 1: Introduction

- What is this tutorial about?
- Overview of Queries/Tasks
- Measures of Success
- Overview of Techniques

Conjunctive Queries

- Common for all queries we are interested in:
  - Joins
  - Projections
  - Selections comparing attributes to constants, e.g., R3.Z2=4

```
SELECT X1, X2        --Projection
FROM R1, R2, R3   --Joined relations
WHERE   R1.Z1 = R2.Z1 AND R1.A2 = R2.A2
        AND R2.X2 = R3.X2
        --Selections
        AND R3.Z2 = 4
```

Conjunctive Queries

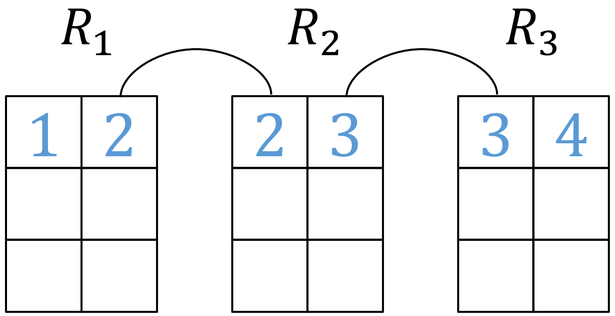
- Common for all queries we are interested in:
  - Joins
  - Projections
  - Selections comparing attributes to constants, e.g., R3.Z2=4

- Formalized by the language of Conjunctive Queries

Relations from the DB   Projected-out column   Selection   Full CQ = No projections

$$Q(x_1, x_2) :- R_1(x_1, z_1), R_2(z_1, x_2), R_3(x_2, 4)$$

Equi-join conditions



$$Q(1, 3) :- R_1(1, 2), R_2(2, 3), R_3(3, 4)$$

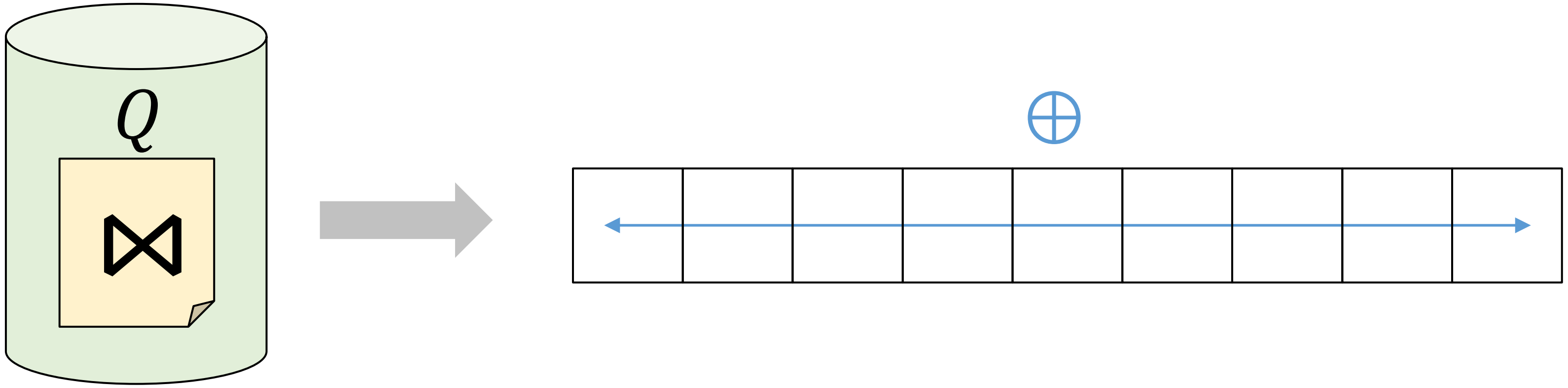
Query Answer

Aggregates

- Task: perform aggregate operations over the join output
- Examples:
  - Count (exactly) the number of query answers
  - Find the lightest 4-path in a graph
  - Find the answer with the highest probability of being true
  - Does the query have any answer at all?

Aggregates

- Task: perform aggregate operations over the join output
- Apply a (binary) operator $\bigoplus$ on some attribute of the join result
- Can be SUM $(+)$, MIN $(\min)$, MAX $(\max)$, BOOLEAN OR $(\vee)$
- More complex:
  - Group-bys
  - …



Speeding up Aggregates

join

R

A	*B*	*w*
1	0	1
2	0	2
3	0	3
…	0	…
n	0	*n*

S

B	*C*	*w*
0	1	1
0	2	2
0	3	3
0	…	…
0	n	*n*

```sql
-----------------------------
-- Query 1
-----------------------------
SELECT min(R.W + S.W) as weight
INTO record1
FROM R, S
WHERE R.B=S.B;
```

DB materializes the join

```sql
-----------------------------
-- Query 2
-----------------------------
SELECT min(W1+W2) as weight
INTO record2
FROM
    (SELECT B, MIN(W) W1
    FROM R
    GROUP BY B) T1,
    (SELECT B, MIN(W) W2
    FROM S
    GROUP BY B) T2
WHERE T1.B = T2.B;
```

Force DB to
not materialize the join

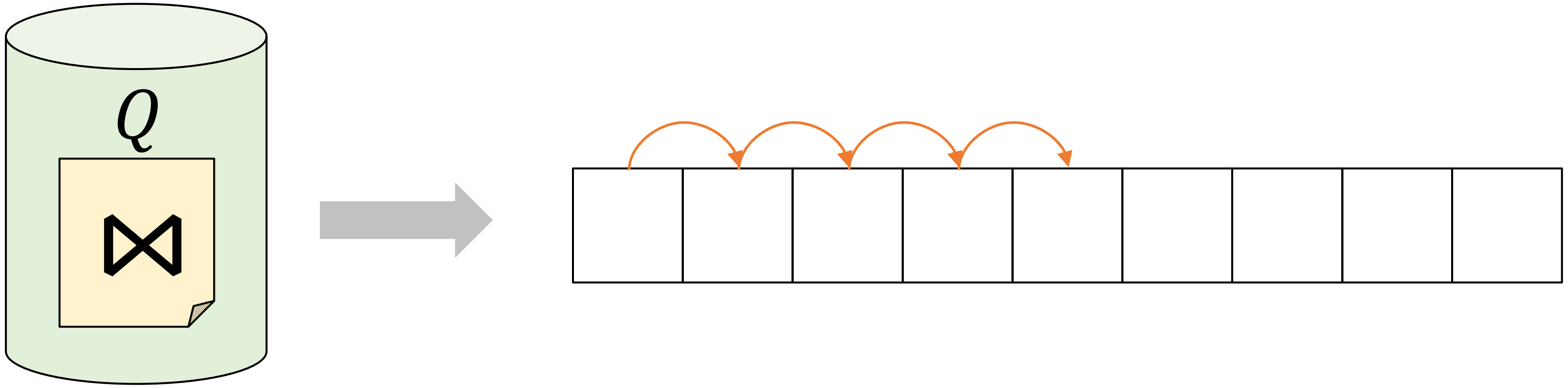
n= 1,000: t_{Q1}= 0.1 sec t_{Q2}<1 msec

n=10,000: t_{Q1}= 9.4 sec t_{Q2}=3 msec

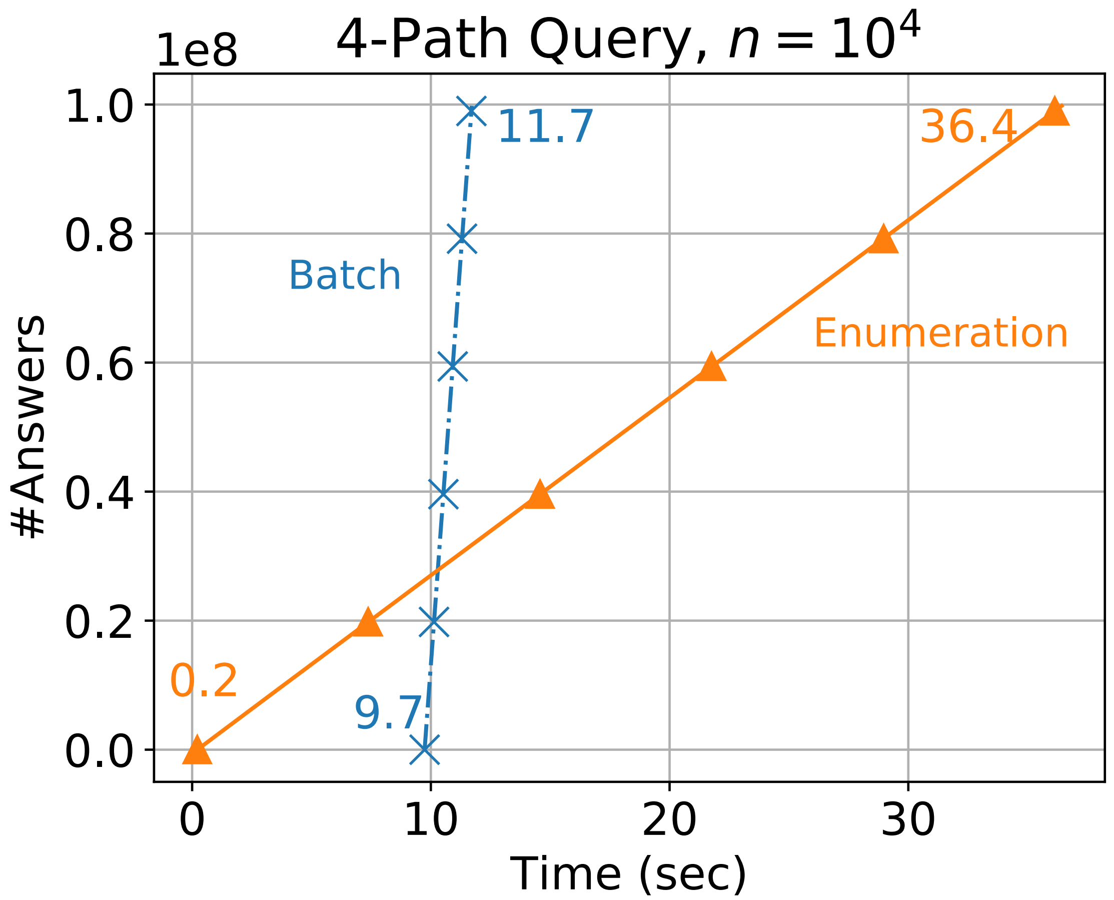
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Unranked Enumeration

- Task: enumerate one-by-one the join output (in arbitrary order)
- Why?
  - The join output can be extremely large
  - Computing it all at once may be hopelessly slow because of its size
  - Enumeration alleviates this by returning *some answers* quickly



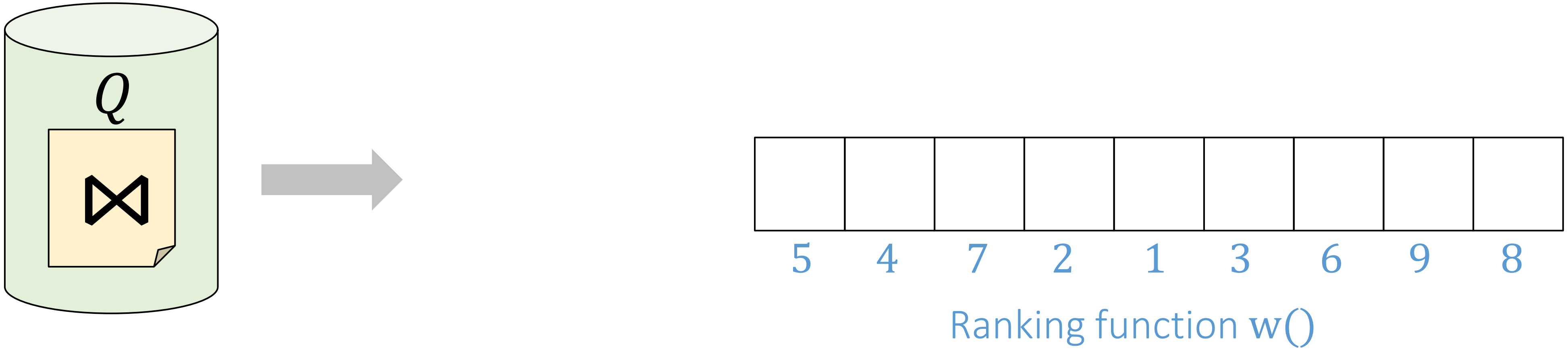
Unranked Enumeration in Practice



Starts returning answers with minimal preprocessing
(only after 0.2 seconds)

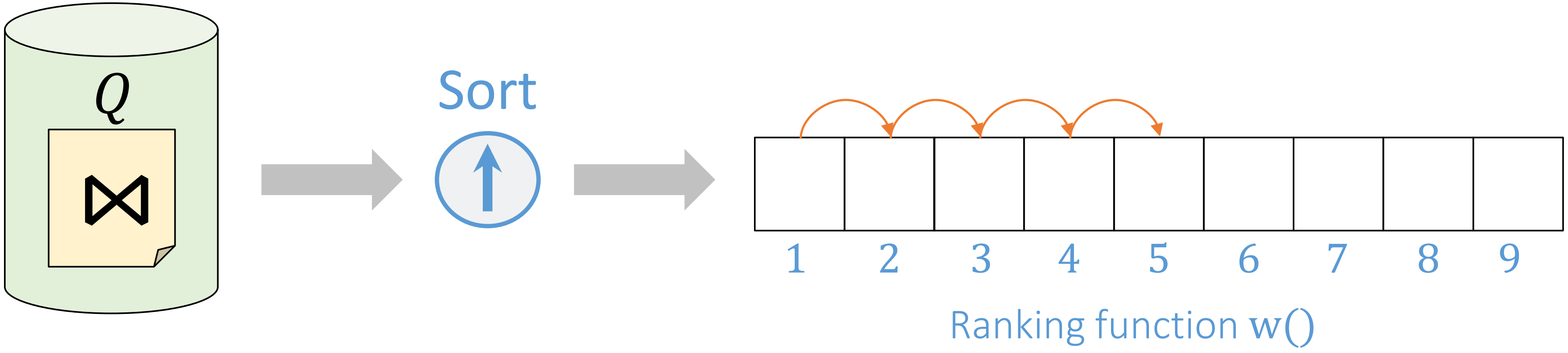
Ranking: Ranked Enumeration

- Task: enumerate the join output in ranked order (according to a ranking function)

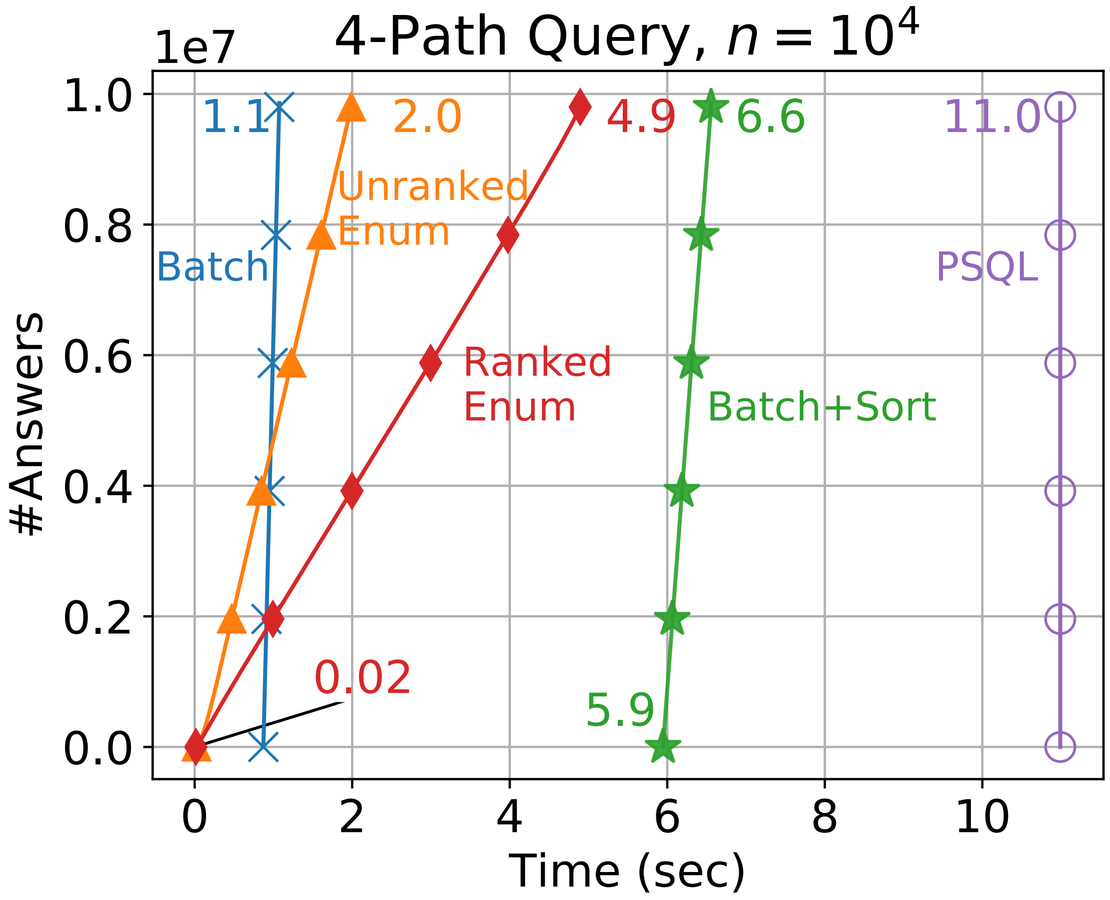


Ranking: Ranked Enumeration

- Task: enumerate the join output in ranked order (according to a ranking function)
- This prioritizes enumeration with some measure of importance
  - This can be freshness (date), quality, trust, or other application-dependent
- The join result is sorted *incrementally*



Ranked Enumeration in Practice



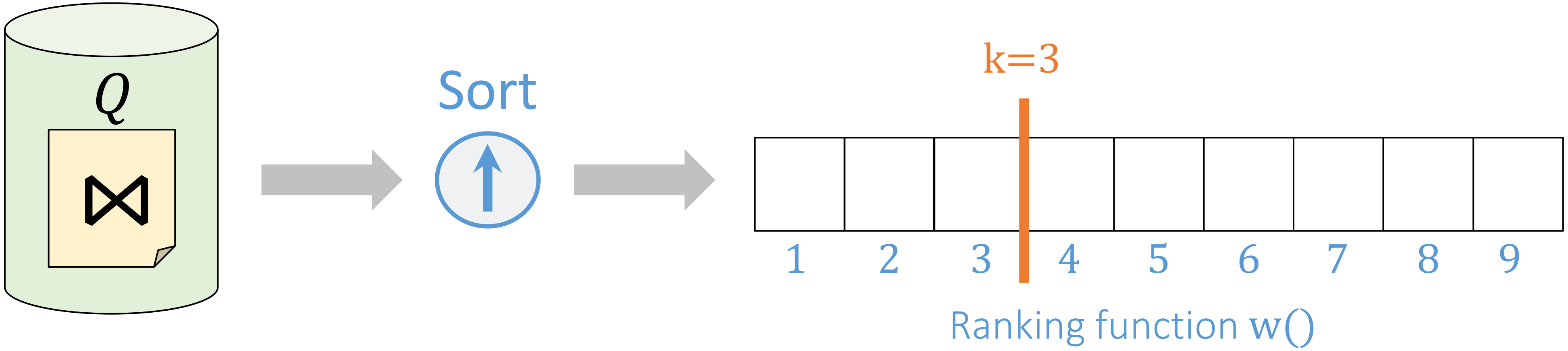
Top results returned very fast by Ranked Enumeration

Only slower than unranked by a factor of 3

Returns the last answer faster than sorting!

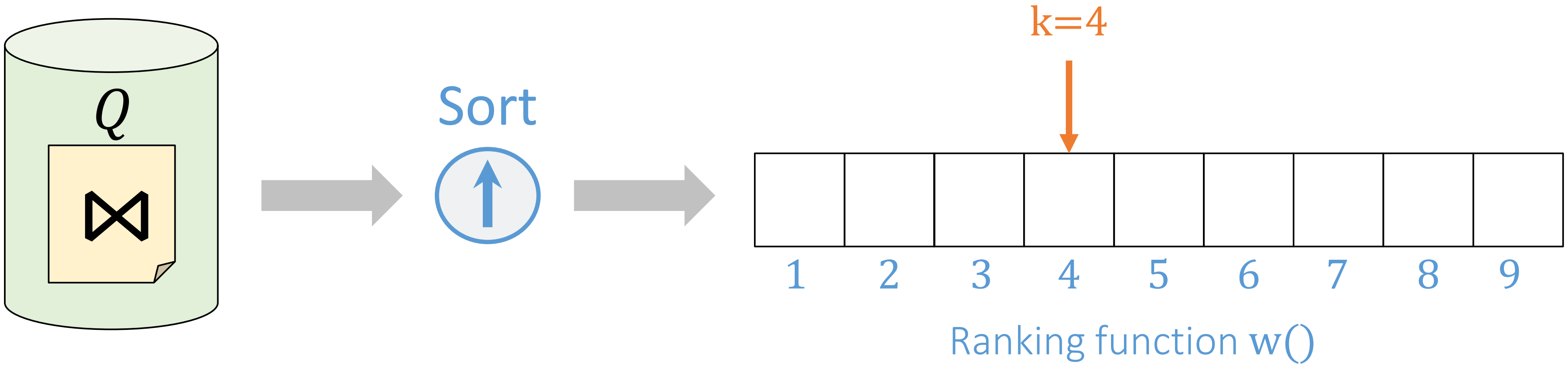
Ranking: Top-k

- Task: top-k join answers (according to a ranking function)
- A special case of ranked enumeration
  - We stop at LIMIT k
  - The k answers don't need to be enumerated



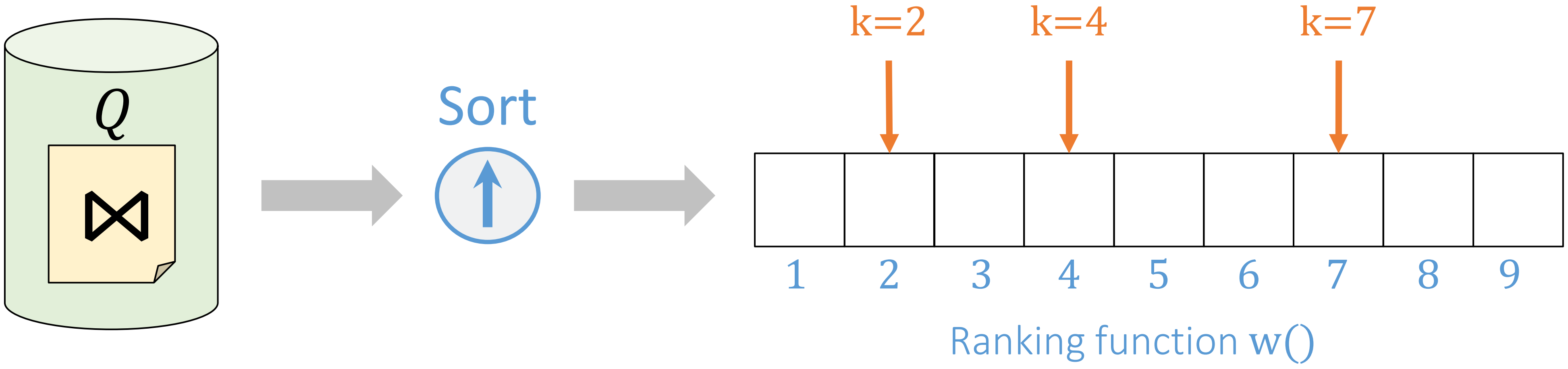
Selection

- Task: return the k^{th}-ranked answer (according to a ranking function)
- Median, quantiles, etc.
- Not to be confused with relational-algebra selection!



Direct Access

- Task: handle multiple selection tasks (according to a ranking function)
- Ideally, a data structure handles multiple accesses more efficiently than performing selection multiple times



Tractable and Non-Tractable Queries

- Tractable selection: in time close to the DB size (not the join output size)
- Tractable direct access: data structure construction time close to the DB size and allows accesses much more efficiently

join

R

A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

S

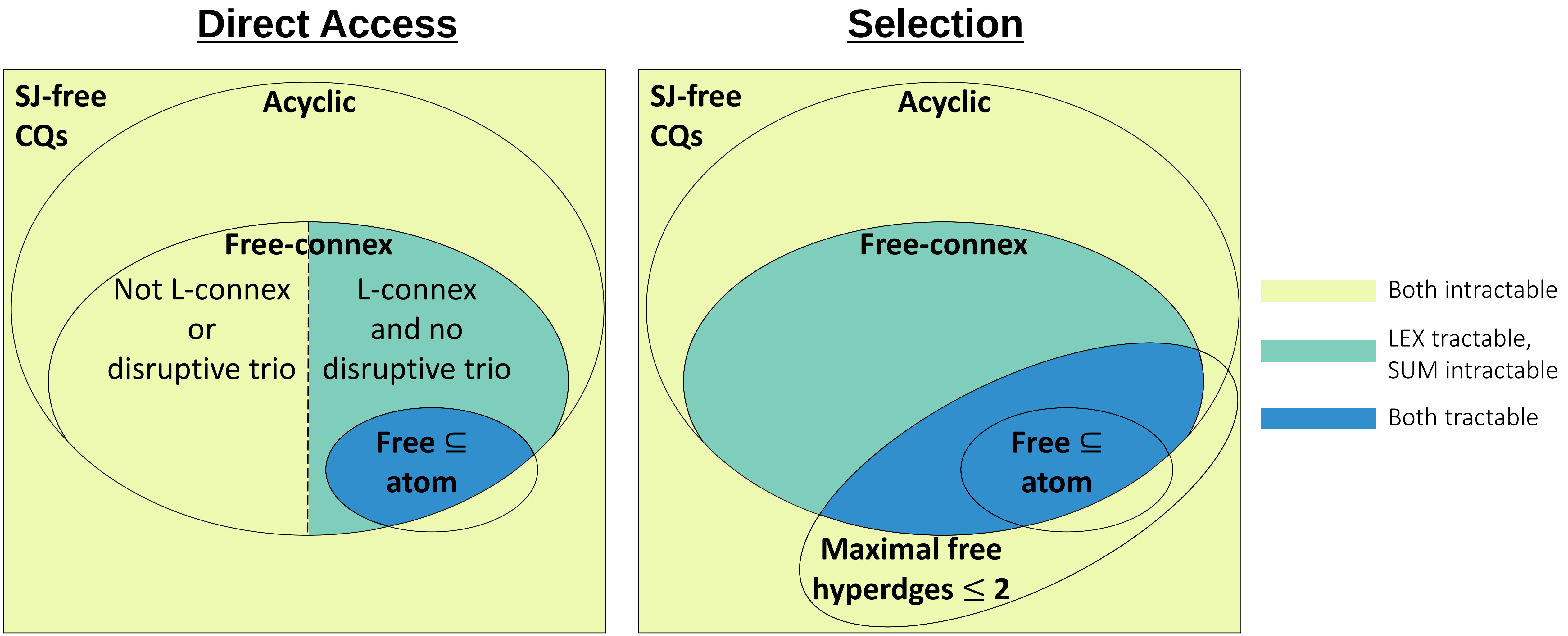
B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

Q(A,B,C) :- R(A,B), S(B,C)

(LEX) $A \to B \to C$, Direct Access	✓
(LEX) $A \to C \to B$, Direct Access	✗
(LEX) $A \to C \to B$, Selection	✓
(SUM) $A + B + C$, Direct Access	✗
(SUM) $A + B + C$, Selection	✓

Carmeli, Tziavelis, Gatterbauer, Kimelfeld, Riedewald. Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries. PODS'21 https://doi.org/10.1145/3452021.3458331
Extended version: https://arxiv.org/abs/1911.05582
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Dichotomies for Direct Access and Selection

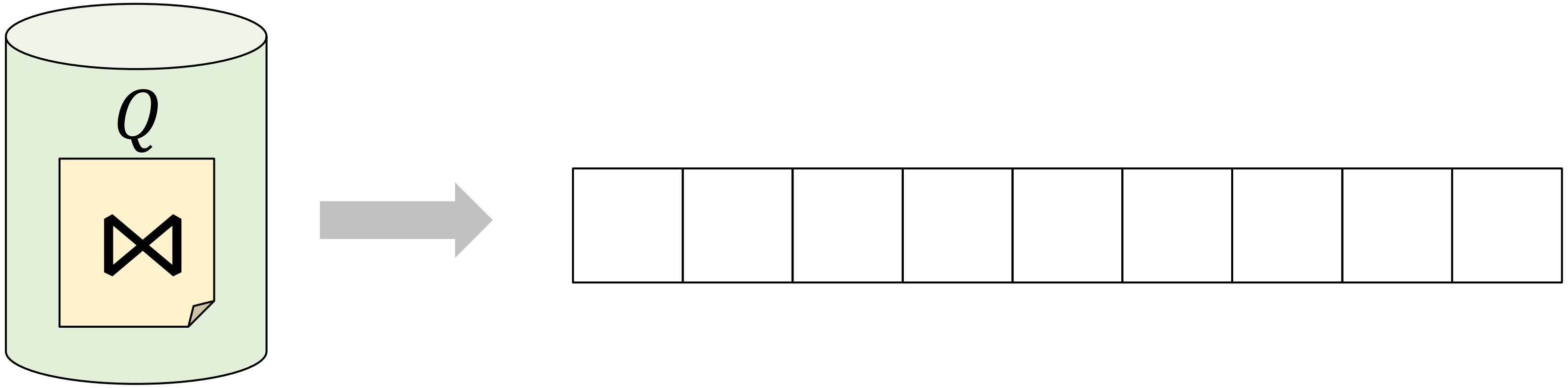


(hardness results conditional on fine-grained complexity hypotheses)

Carmeli, Tziavelis, Gatterbauer, Kimelfeld, Riedewald. Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries. PODS'21 https://doi.org/10.1145/3452021.3458331
Extended version: https://arxiv.org/abs/1911.05582
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Batch

- Task: Simply produce the join output (no task on top of join)
- This is also part of the tasks that we consider (f is the identity function), but there is obviously no way to avoid the join materializing cost
- Still, the goal is to avoid unnecessary intermediate results



Avoiding Intermediate Results in Practice

R join *S* join

A	*B*
n	-1
2	-1
1	-1
0	0

B	*C*
-1	1
-1	2
-1	3
0	0
1	-2
2	-2
n	-2

C	*D*
0	0
-2	1
-2	2
-2	n

```
---------------------------
-- Query 1
---------------------------
select *
into record1
from R natural join S natural join T;
```

```
---------------------------
-- Query 2
---------------------------
With S2 as
    (SELECT *
    FROM S
    WHERE S.B in
        (SELECT R.B
        FROM R)),
S3 as
    (SELECT *
    FROM S2
    WHERE S2.C in
        (SELECT T.C
        FROM T))
select a, b, c, d
into record2
from R natural join S3 natural join T;
```

n=1,000: t_{Q1}=1.4 sec t_{Q2}=5 msec

n=2,000: t_{Q1}=6.1 sec $O(n^2)$ ☹ t_{Q2}=8 msec $O(n)$ ☺

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Outline Part 1

Part 1: Introduction

- What is this tutorial about?
- Overview of Queries/Tasks
- Measures of Success
- Overview of Techniques

Complexity Notation

- Standard O and Ω notation for time and memory complexity in the RAM model of computation
  - Database size n
  - Query size ℓ
  - Output size r
- We present most results using data complexity
  - Scalability in data size
  - Treat query size ℓ as a constant
  - E.g., $\mathrm{O}\big(f(\ell) \cdot n^{f(\ell)} + (\log n)^{f(\ell)} \cdot r\big)$ simplifies to $\mathrm{O}\big(n^{f(\ell)} + (\log n)^{f(\ell)} \cdot r\big)$
- $\tilde{\mathrm{O}}$-notation (soft-O): abstracts away polylog factors that clutter formulas
  - E.g., $\mathrm{O}\big(n^{f(\ell)} + (\log n)^{f(\ell)} \cdot r\big)$ further simplifies to $\tilde{\mathrm{O}}\big(n^{f(\ell)} + r\big)$

Measures of Success

- A join of ℓ relations can have an $O(n^\ell)$ output
- Ideally, we want running times close to $O(n)$

- Assumptions:
  - No pre-computed data structures such as indexes, sorted representation, materialized views
  - In-memory computation, no I/O cost
  - Hash tables support $O(1)$ lookups, otherwise additional log factor in analysis

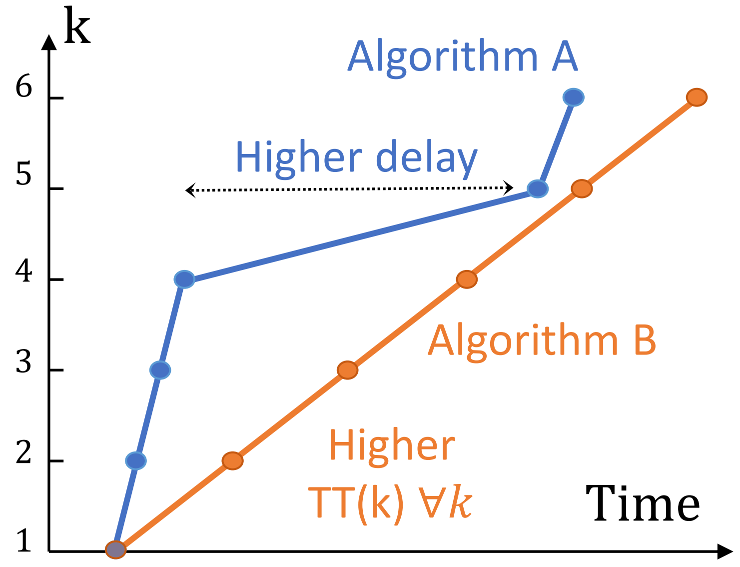
- Linear time $O(n)$ is basically "for free" since we must look at each input tuple at least once for all the problems we discuss

Enumeration: TT(k) vs Delay

- Two ways to measure success for an enumeration algorithm
  - $\mathrm{TT}(k)$: Time-to-the-k$^{\mathrm{th}}$ answer
  - Preprocessing time and delay between answers

- Why delay?
  - Can be useful for bounding $\mathrm{TT}(k)$
  - If delay $\leq c$ then $\mathrm{TT}(k) \leq |\mathrm{Prep}| + ck$

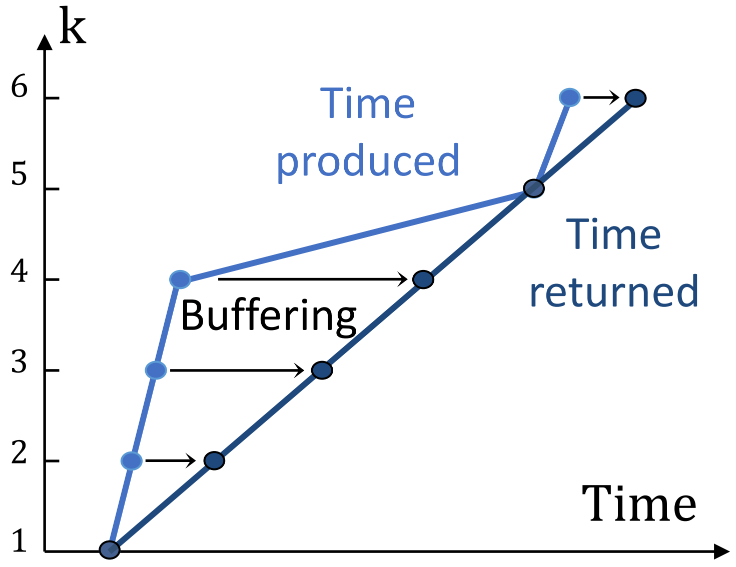
- Low delay is sufficient but not necessary for low $\mathrm{TT}(k)$



Lower max delay does not necessarily give a faster algorithm!

Enumeration: TT(k) vs Delay

- Given a low-$\mathrm{TT}(k)$ algorithm, can we lower its delay?
  - Yes! By buffering the answers and returning them at regular intervals
  - But this slows down the algorithm…
- When is a low-delay algorithm desirable?
  - A downstream application requires regular interarrival times
  - Additionally, there is not enough buffer space



Delay is a more restrictive requirement than $\mathbf{TT}(k)$
but with limited practical applications

Capelli, Strozecki. Incremental delay enumeration: space and time. Discrete Applied Mathematics 2019. https://doi.org/10.1016/j.dam.2018.06.038
Deep, Hu, Koutris. Enumeration Algorithms for Conjunctive Queries with Projection. ICDT'21. https://doi.org/10.4230/LIPIcs.ICDT.2021.14
Carmeli, Kröll. On the enumeration complexity of unions of conjunctive queries. TODS 2021. https://doi.org/10.1145/3450263
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Outline Part 1

Part 1: Introduction

- What is this tutorial about?
- Overview of Queries/Tasks
- Measures of Success
- Overview of Techniques

Factorized Representations

To be continued in Part 4&7

- What is the key idea?
  - Represent the query answers compactly in a lossless way
  - Allow other tasks directly on this representation
  - Forms the basis for all the tasks that we discuss (enumeration, direct access, etc.)

- Why "factorized"?
  - Factorization is the process of simplifying a formula by identifying common subexpressions
  - Similarly, query answers can have redundancy which can be "factored out"

Common factor

$$ab + ac = a(b + c)$$

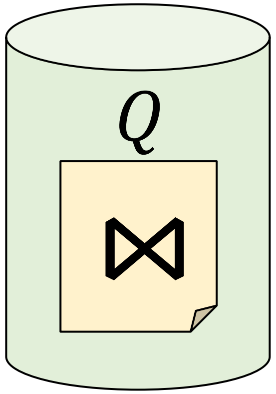
4 elements,
3 operations

3 elements,
2 operations

Factorized Representations

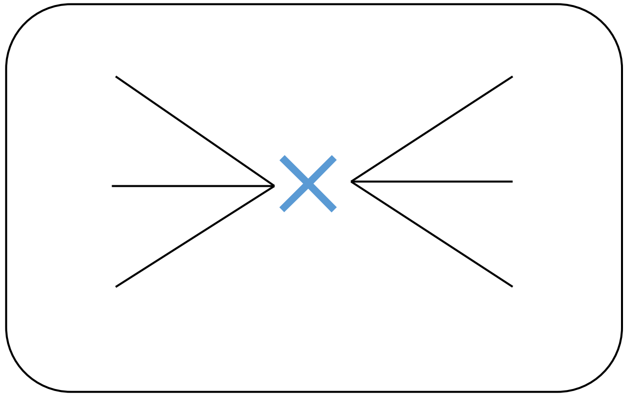
To be continued in Part 4&7

DB size: $O(n)$

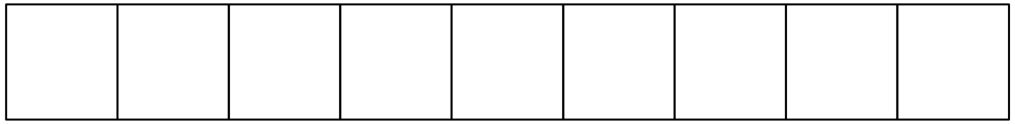


Constructed directly from the DB

Factorized Representation



Lossless



Size $\ll$ Join-output size

$O(n)$ size for equi-joins
$O(n \text{ polylog } n)$ for inequality-joins
$O(n^2)$ for theta-joins

n = #tuples
ℓ = #relations

Join-output size: $O(n^\ell)$

Olteanu, Závodný. Size bounds for factorised representations of query results. TODS 2015 https://doi.org/10.1145/2656335

Tziavelis, Gatterbauer, Riedewald. Beyond Equi-joins: Ranking, Enumeration and Factorization. PVLDB'21 https://doi.org/10.14778/3476249.3476306

Acyclic queries and the Yannakakis Algorithm

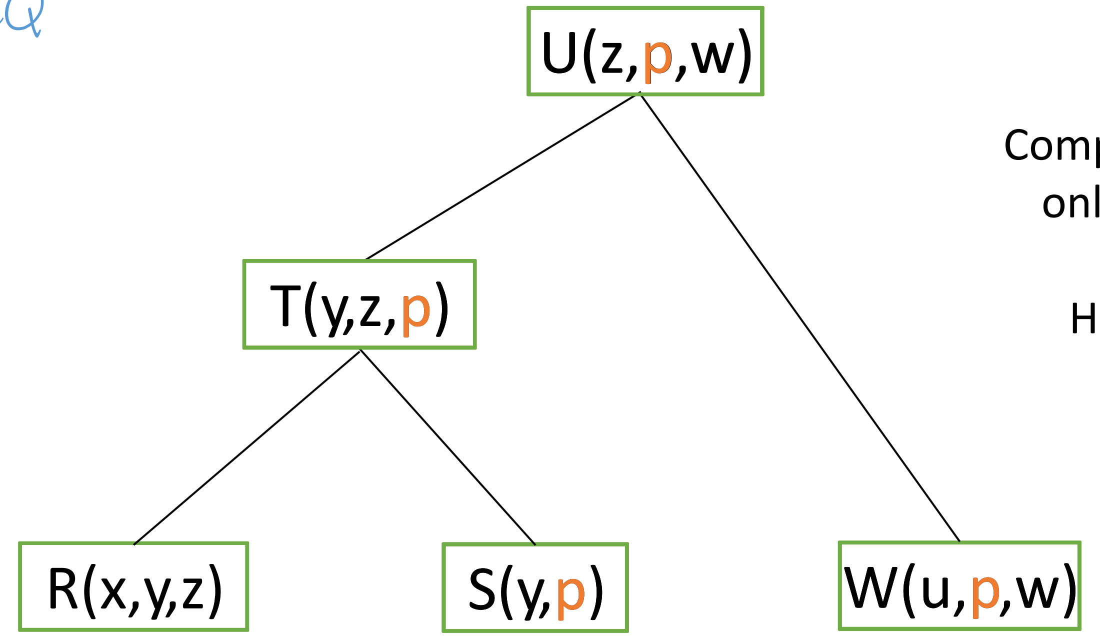
To be continued in Part 3

- What is the key idea?
  - For acyclic queries (that do not require cyclic joins), we can remove in linear time all dangling tuples: those that are not part of any answer
  - This allows us to evaluate them very efficiently
  - The Yannakakis algorithm answers acyclic CQs in $O(n+r)$, which is optimal

How do we know whether a CQ does not require cyclic joins?

Join Tree

- Nodes: relations
- the nodes containing the same variable are connected



Compared to query plans: only partial join order.

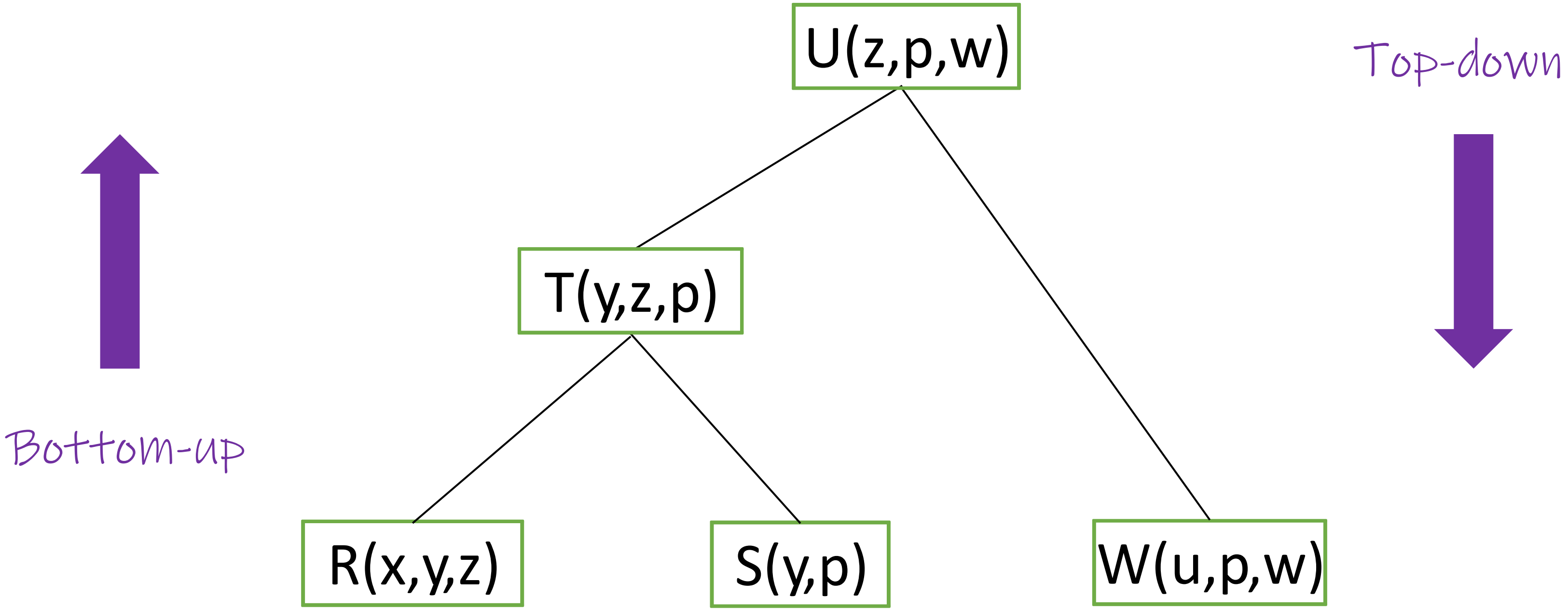
Here T⋈R and T⋈S before T⋈U.

Acyclic queries and the Yannakakis Algorithm

To be continued in Part 3

- Yannakakis algorithm: two passes over the database according to the join tree order
- This removes all dangling tuples
- Consequently, joins (following the tree order) can never produce intermediate results that are not needed

$O(n + r)$



Yannakakis. Algorithms for acyclic database schemes. VLDB'81 https://dl.acm.org/doi/10.5555/1286831.1286840
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Projections and free-connex queries

To be continued in Part 3

- What is the key idea?
  - With projections, it is more difficult to avoid unnecessary intermediate results
  - For Yannakakis, as well as enumeration and direct access, there are "easy" and "hard" projections.
    The easy cases are captured by the class of free-connex CQs
  - For free-connex CQs, it is possible to eliminate all projections in linear time

Projected-out

Q(y,z,u) :- R(x,y), S(y,z), T(z,u) → $\pi_y(R) \bowtie S \bowtie T$

Q(z,u) :- R(x,y), S(y,z), T(z,u) → $\cancel{\pi_z(R \bowtie S) \bowtie T}$

y is needed for $R \bowtie S$

$R \bowtie S$ can take $O(n^2)$

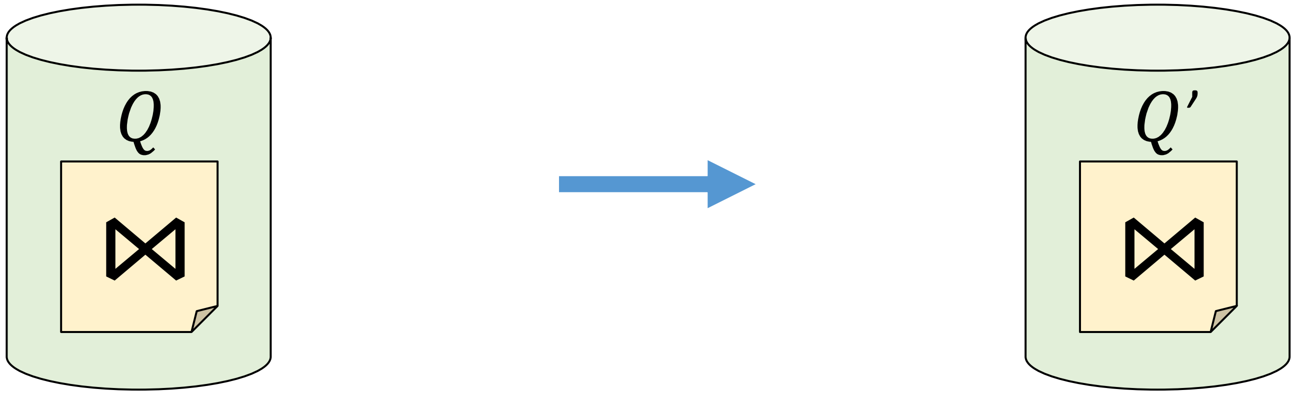
Bagan, Durand, Grandjean. On acyclic conjunctive queries and constant delay enumeration. CSL'07. https://doi.org/10.1007/978-3-540-74915-8_18
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Projections and free-connex queries

To be continued in Part 3

Database D
size $O(n)$

Database D'
size $O(n)$



Q has projections
and is free-connex

Q' has no projections

Free-connex	Q(y,z,u) :- R(x,y), S(y,z), T(z,u)	→	$\pi_y(R) \bowtie S \bowtie T$
Free-connex	Q(z,u) :- R(x,y), S(y,z), T(z,u)	$\xrightarrow{O(n)}$	Q'(z,u) :- T'(z,u)
Not free-connex	Q(x,u) :- R(x,y), S(y,z), T(z,u)	→	✘

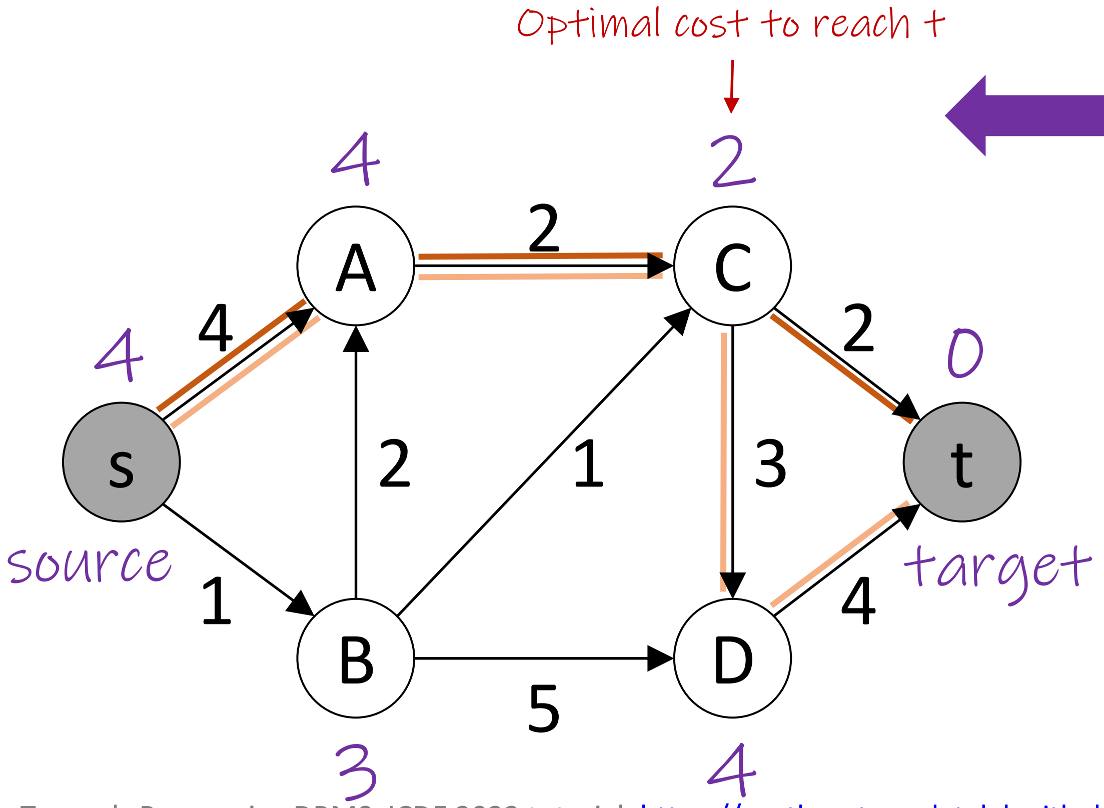
(conditional on fine-grained complexity hypotheses)

Bagan, Durand, Grandjean. On acyclic conjunctive queries and constant delay enumeration. CSL'07. https://doi.org/10.1007/978-3-540-74915-8_18
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Dynamic Programming

To be continued in Part 5

- Dynamic Programming (DP) is the archetypical paradigm for solving problems that exhibit a shared structure
- Bellman-Ford algorithm: shortest path in a DAG G in $O(|G|)$



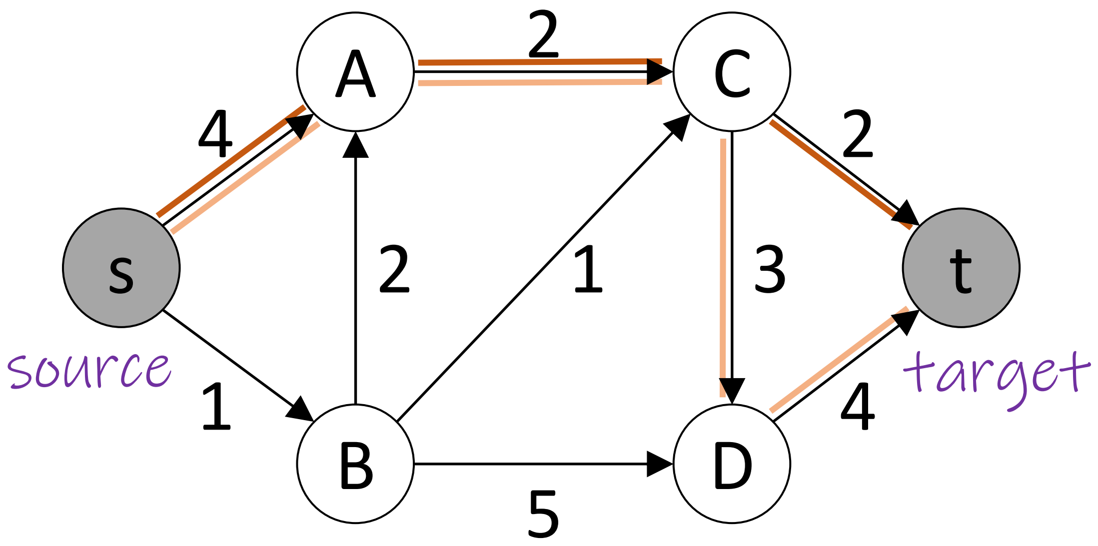
Bottom-up: from target node to source node (in reverse topological sort)

Shared structure
different paths share common edges

Dynamic Programming

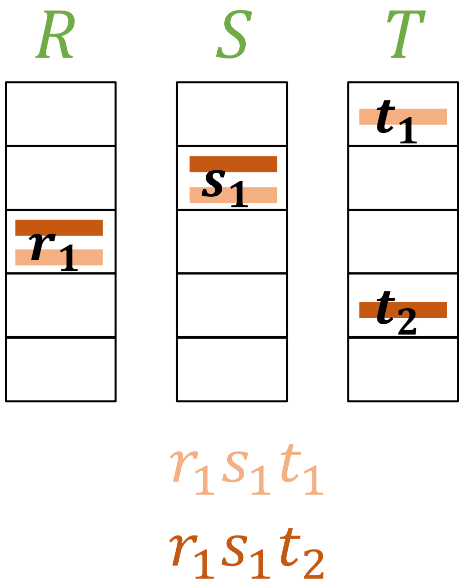
To be continued in Part 5

- What is the key idea?
  - We can use DP to compute different aggregates over the join result
  - The factorized representation gives the shared structure!



Shared structure
different paths share common edges

different output tuples share common input tuples



Semirings and DP

To be continued in Part 5

- Using DP on acyclic queries we can compute in linear time:
  - The exact count of query answers $+, \times$
  - The query answer with the minimum sum of weights $\min, +$
  - The query answer with the highest probability of being true $\max, \times$
- These are computed with the same DP algorithm, simply by swapping operators
- What do they have in common? Similar algebraic properties, described by algebraic structures called semirings

Mohri. Semiring frameworks and algorithms for shortest-distance problems. Journal of Automata, Languages and Combinatorics 2002. https://doi.org/10.5555/639508.639512

Abo Khamis, Ngo, Rudra. FAQ: Questions Asked Frequently. PODS'16. https://doi.org/10.1145/2902251.2902280

Any-k

To be continued in Part 6

- Any-k algorithms perform ranked enumeration (related to top-k)
- What is the key idea?
  - The top-1 problem is solvable by DP
  - For any-k, we want the 2^{nd} best, 3^{rd} best,… solution to a DP problem (ranked enumeration for DP)
- There are two incomparable algorithms:
  - One is faster for small k
  - The other is faster for large k
- Latest result: best-of-both-worlds algorithm
- They can be used for any DP problem:
  - Longest increasing subsequence
  - Knapsack
  - …

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB'20 https://doi.org/10.14778/3397230.3397250 Extended report: https://arxiv.org/abs/1911.05582
Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

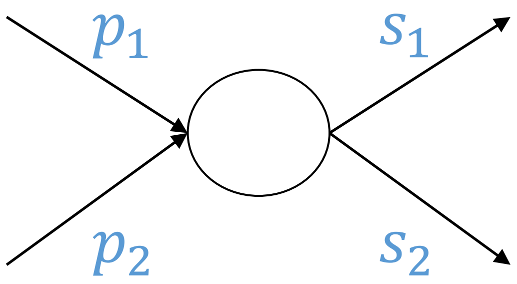
Any-k

To be continued in Part 6

- For appropriate ranking functions, ranked enumeration for CQs is slower than unranked enumeration only by a logarithmic factor
  - Unranked: $\mathrm{TT}(k) = O(n + k)$
  - Ranked: $\mathrm{TT}(k) = O(n + k \log k)$

(free-connex CQs, data complexity, sum-of-weights ranking)

- When the last answer is returned, any-k can be faster than (generic comparison-based) sorting!
  - The query answers are not independent because of their shared (factorized) structure (in combined complexity)



$$s_1 \leq s_2$$

$$\Downarrow$$

$$p_1 + s_1 \leq p_1 + s_2$$

$$p_2 + s_1 \leq p_2 + s_2$$

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB'20 https://doi.org/10.14778/3397230.3397250 Extended report: https://arxiv.org/abs/1911.05582

Tziavelis, Gatterbauer, Riedewald. Any-k Algorithms for Enumerating Ranked Answers to Conjunctive Queries. arXiv'22 https://arxiv.org/abs/2205.05649

Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Worst-case optimal joins

To be continued in Part 2

- What is the key idea?
  - Traditional binary-join plans are suboptimal for cyclic queries because they can take more time than the worst-case output
  - For every query, we can find its worst-case output by solving a linear program, now known as the AGM bound
  - There are WCOJ algorithms that match this bound

Triangle Query

Q(x,y,z) :- R(x,y), S(y,z), T(z,x)

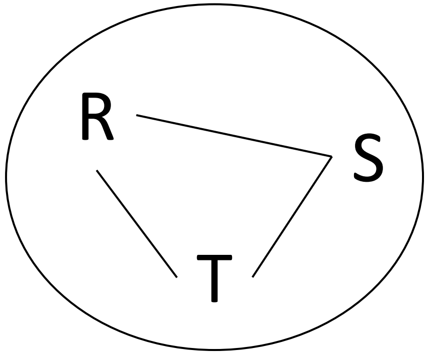
R⋈(S⋈T)

(R⋈S)⋈T

(R⋈T)⋈S

$O(n^2)$

WCOJ



$O(n^{1.5})$

Atserias, Grohe, Marx. Size bounds and query plans for relational joins. SIAM Journal on Computing. 2013. https://doi.org/10.1137/110859440

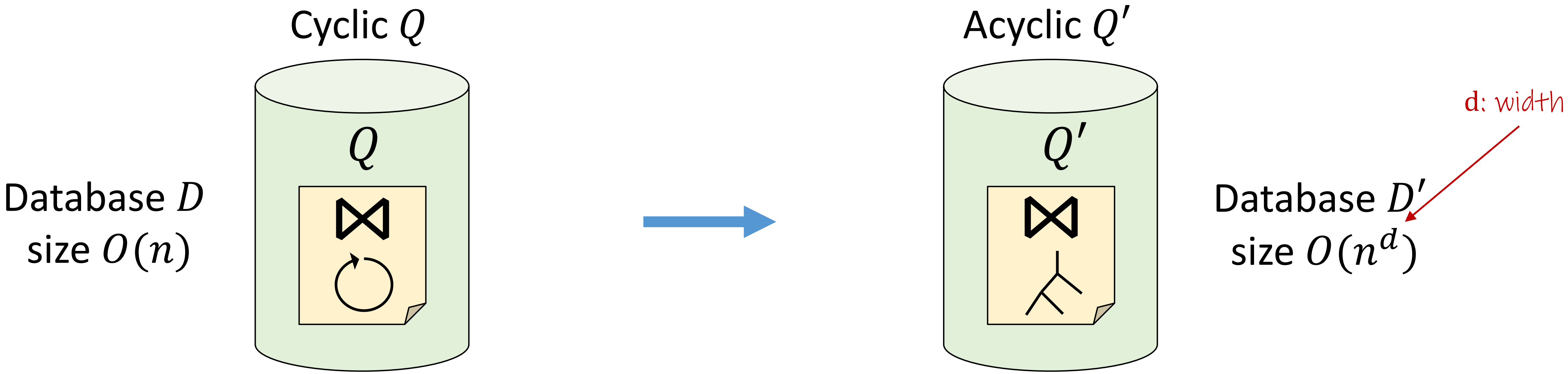
Ngo, Porat, Ré, Rudra. Worst-case optimal join algorithms. JACM 2018. https://doi.org/10.1145/3180143

Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

(Hyper)tree decompositions

To be continued in Part 2

- What is the key idea?
  - Tree decompositions allow us to transform a cyclic query to (potentially a union of) acyclic queries
  - The new query is over larger relations that we have to materialize
  - Cost of materializing those relations => “width” of the decomposition
  - Tree decompositions use WCOJ as a subroutine (for materializing the new relations)



Gottlob, Greco, Leone, Scarcello. Hypertree decompositions: Questions and answers. PODS’16. https://doi.org/10.1145/2902251.2902309
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial