

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

Nikolaos Tziavelis, Wolfgang Gatterbauer, Mirek Riedewald

Northeastern University, Boston

Part 4: Factorization

Northeastern University
Khoury College of Computer Sciences

DATALAB @Northeastern

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

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

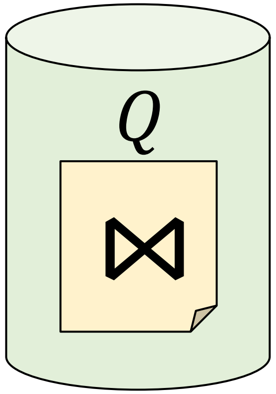
Outline Part 4

Part 4: Factorization

- High-level idea
- Factorized representation of path-CQ
- Factorized representation of tree-CQ & enumeration
- Tuple-level vs Attribute-level representations

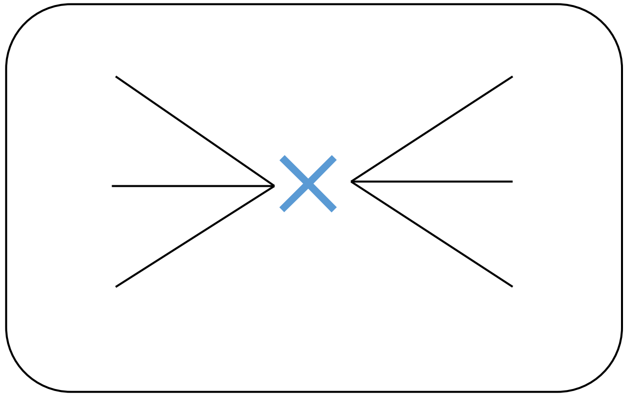
Overview

DB size: $O(n)$



Constructed directly from the DB

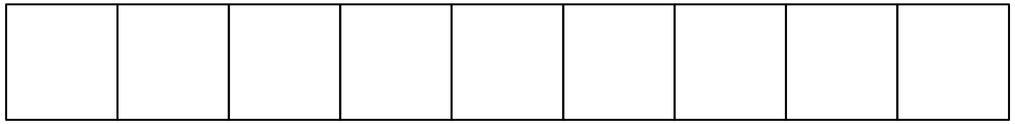
Factorized Representation



Size ≪ Join-output size

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

Lossless



n = #tuples
ℓ = #relations

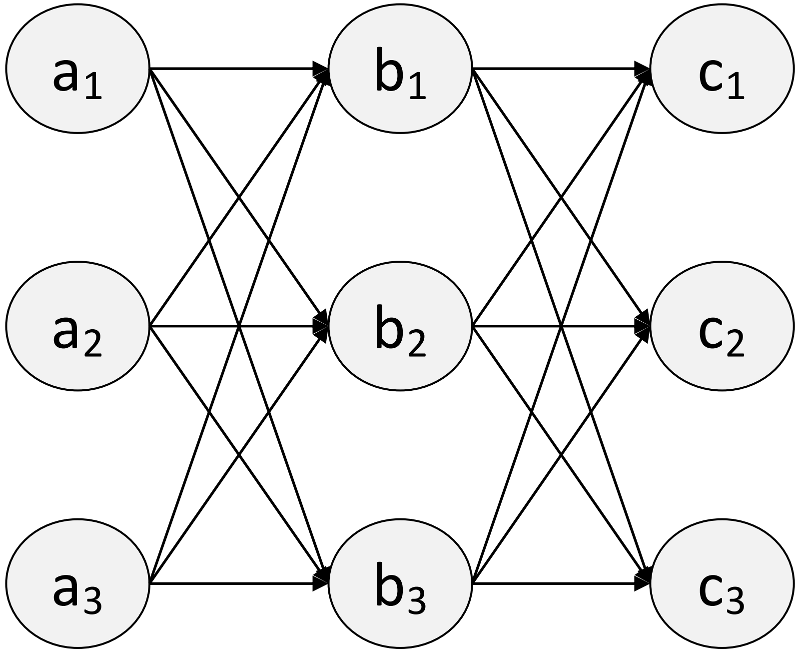
Join-output
size: $O(n^\ell)$

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

Intuition: Edges -> Paths

- How is it possible to have such a compact representation?

Graph



18 edges + 9 nodes

27 paths × 3 nodes

Left-to-right traversals

Paths (from left to right)

$a_1b_1c_1$	$a_2b_1c_1$	$a_3b_1c_1$
$a_1b_1c_2$	$a_2b_1c_2$	$a_3b_1c_2$
$a_1b_1c_3$	$a_2b_1c_3$	$a_3b_1c_3$
$a_1b_2c_1$	$a_2b_2c_1$	$a_3b_2c_1$
$a_1b_2c_2$	$a_2b_2c_2$	$a_3b_2c_2$
$a_1b_2c_3$	$a_2b_2c_3$	$a_3b_2c_3$
$a_1b_3c_1$	$a_2b_3c_1$	$a_3b_3c_1$
$a_1b_3c_2$	$a_2b_3c_2$	$a_3b_3c_2$
$a_1b_3c_3$	$a_2b_3c_3$	$a_3b_3c_3$

Intuition: Paths -> Edges

- How is it possible to have such a compact representation?
  - Because of shared structure (redundancy)

(exponentially) more compact ✓
lossless ✓

Paths (from left to right)

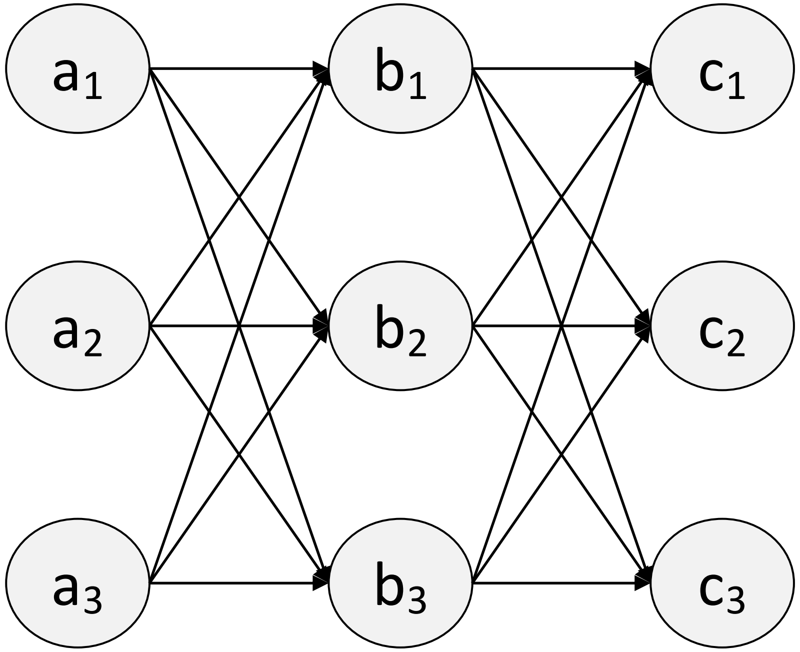
27 paths × 3 nodes

$a_1b_1c_1$	$a_2b_1c_1$	$a_3b_1c_1$
$a_1b_1c_2$	$a_2b_1c_2$	$a_3b_1c_2$
$a_1b_1c_3$	$a_2b_1c_3$	$a_3b_1c_3$
$a_1b_2c_1$	$a_2b_2c_1$	$a_3b_2c_1$
$a_1b_2c_2$	$a_2b_2c_2$	$a_3b_2c_2$
$a_1b_2c_3$	$a_2b_2c_3$	$a_3b_2c_3$
$a_1b_3c_1$	$a_2b_3c_1$	$a_3b_3c_1$
$a_1b_3c_2$	$a_2b_3c_2$	$a_3b_3c_2$
$a_1b_3c_3$	$a_2b_3c_3$	$a_3b_3c_3$

Factorization

18 edges + 9 nodes

Graph



Relationship to Algebraic Factorization

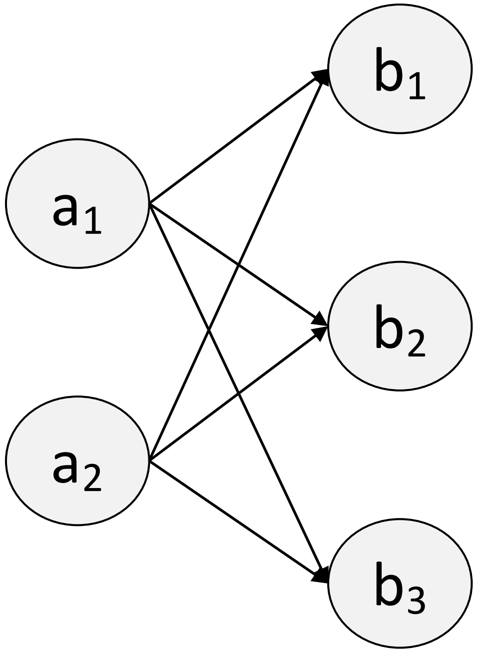
- Factorization of algebraic formulas can also be interpreted in this way

Distributivity of × over +

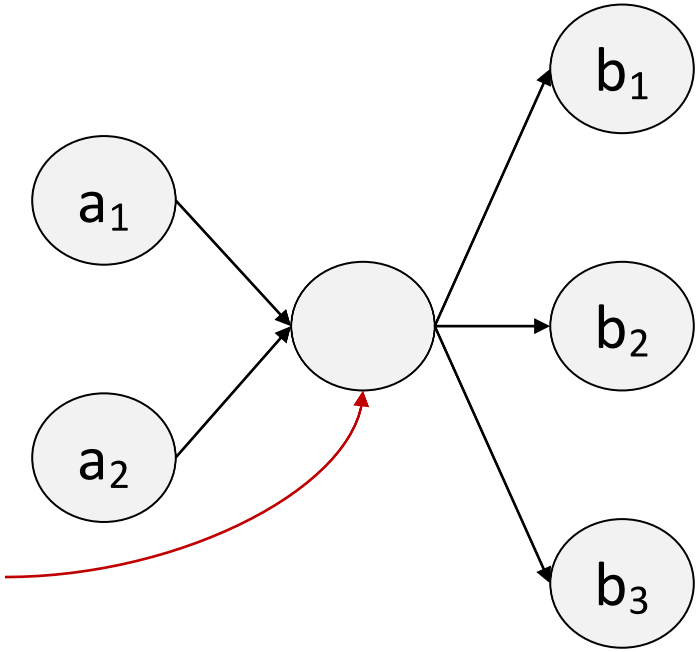
$(a_1 \times b_1) + (a_1 \times b_2) + (a_1 \times b_3) +$
$(a_2 \times b_1) + (a_2 \times b_2) + (a_2 \times b_3)$

Factorization

$(a_1 + a_2) \times (b_1 + b_2 + b_3)$



Node in the middle forces paths to share edges



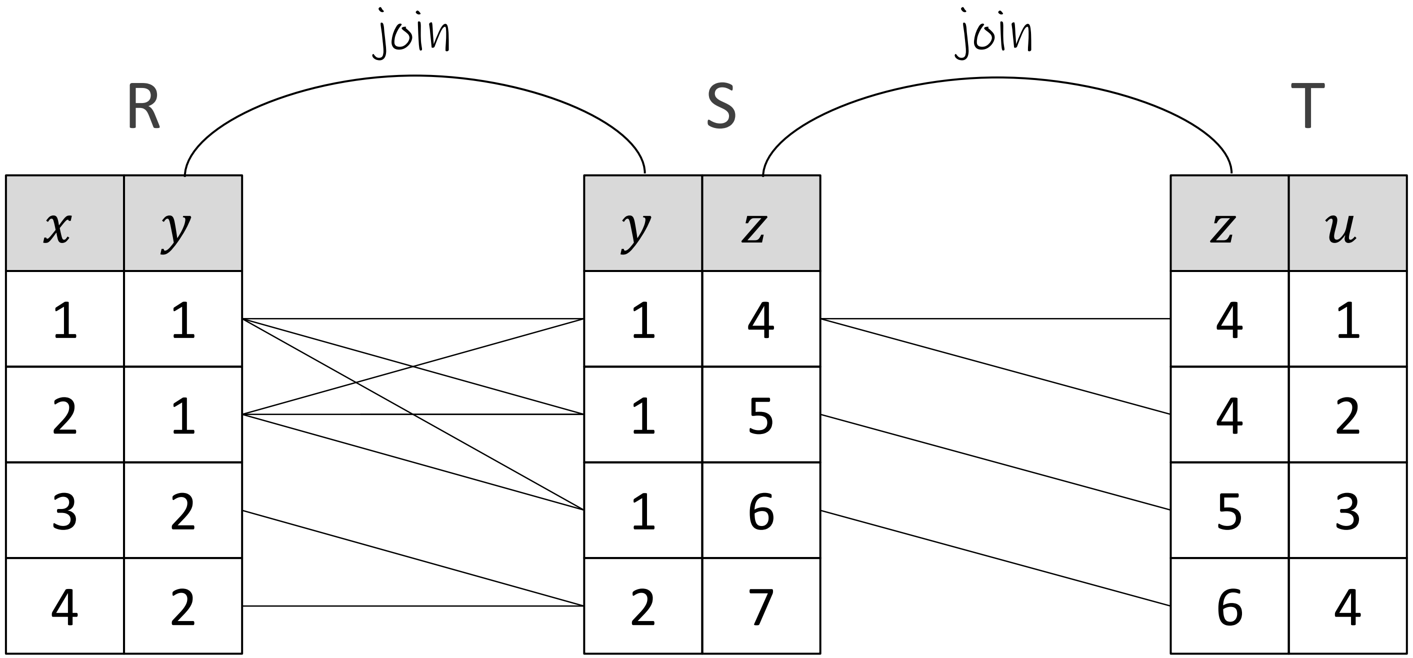
Outline Part 4

Part 4: Factorization

- High-level idea
- Factorized representation of path-CQ
- Factorized representation of tree-CQ & enumeration
- Tuple-level vs Attribute-level representations

Factorized Representation of Join Query Results

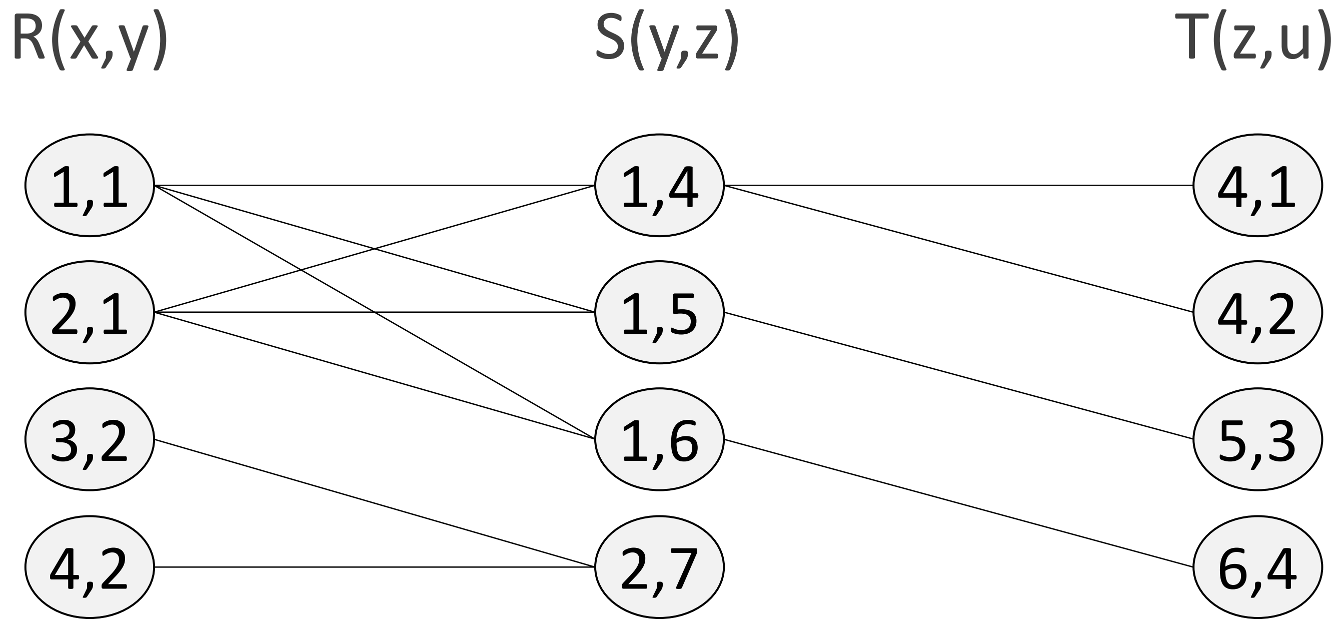
Q(x,y,z,u) :- R(x,y), S(y,z), T(z,u)



Connections: joining tuples

Factorized Representation of Join Query Results

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

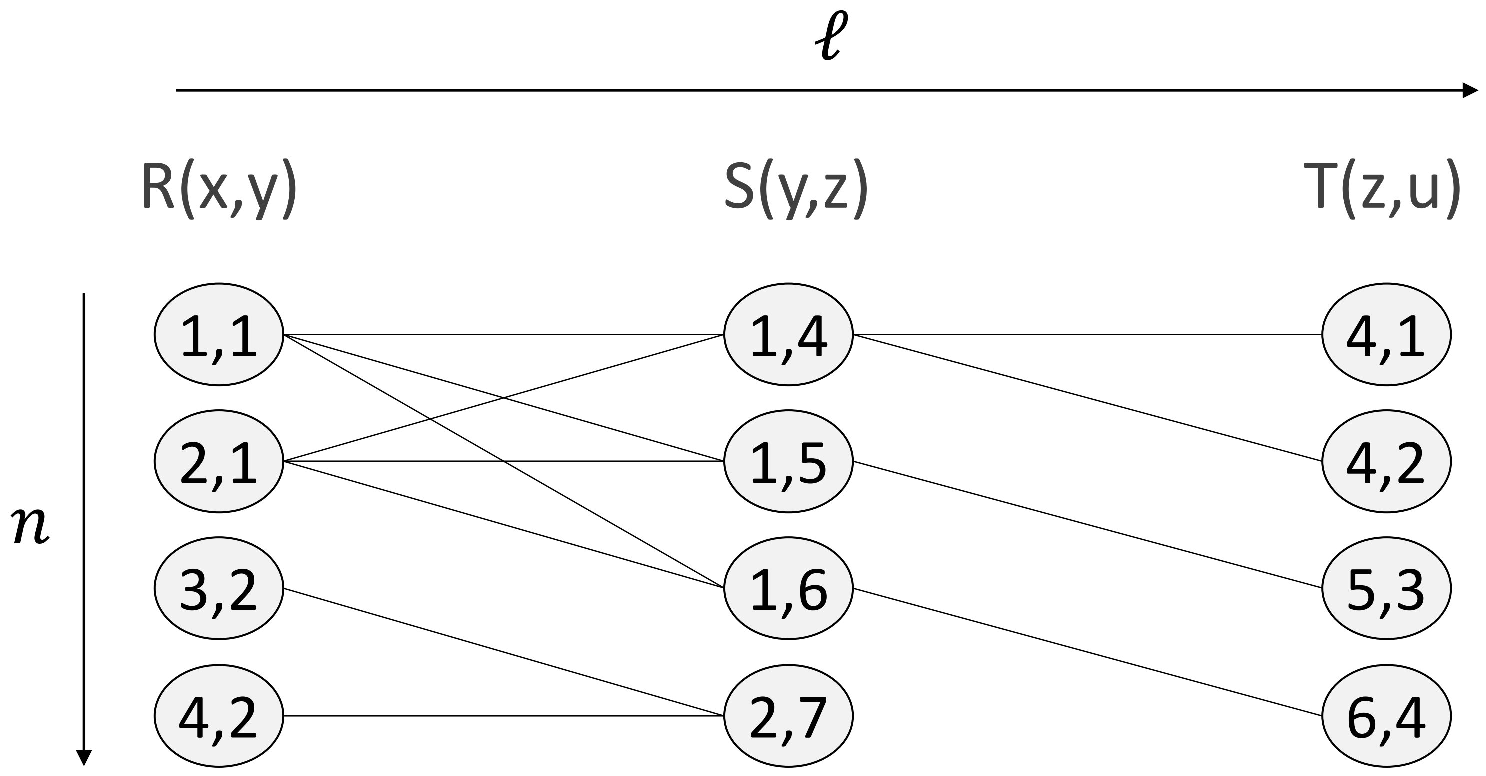


Graph Representation

- Nodes = Tuples
- Edges = Joining pairs
- Paths = Join results

Factorized Representation of Join Query Results

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

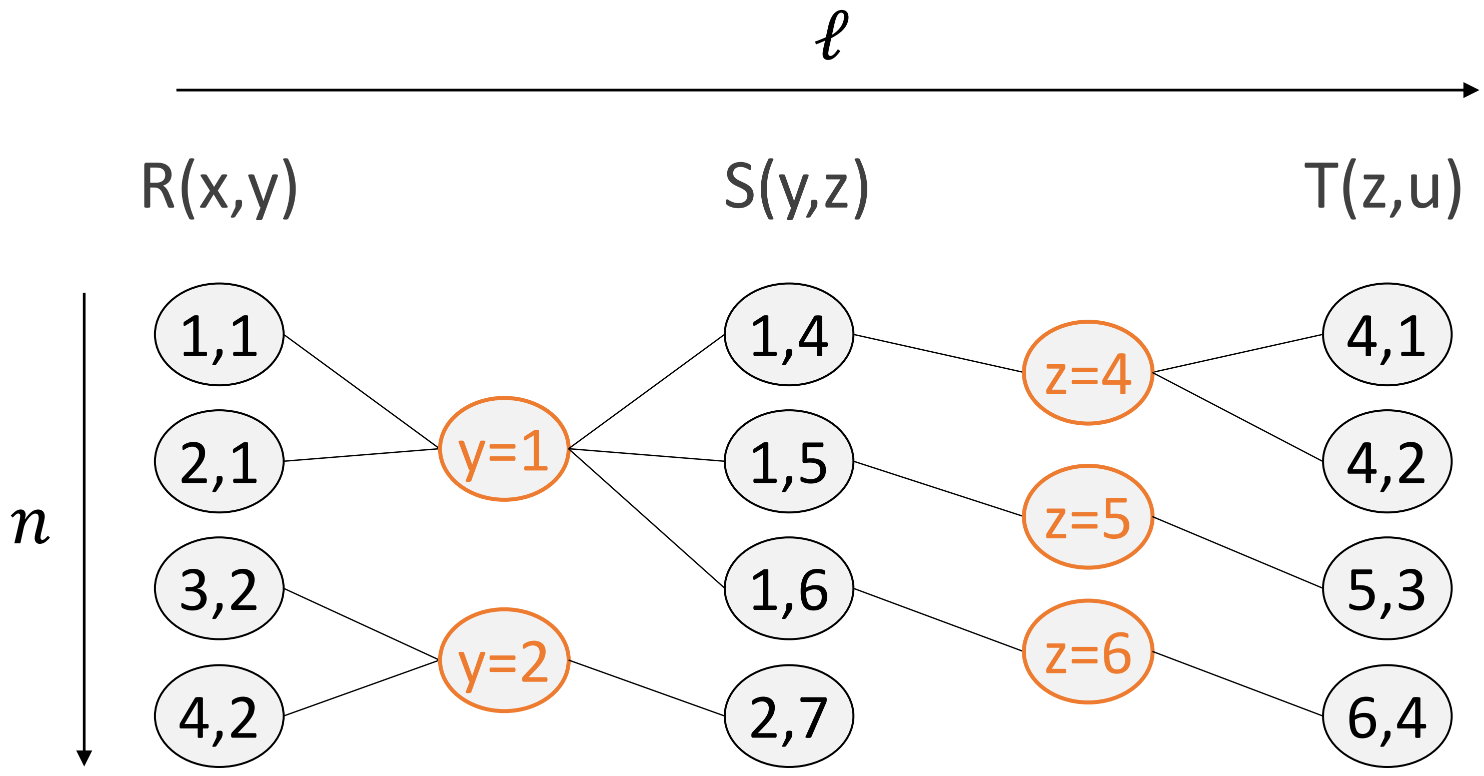


- Can we lower the quadratic cost?
- If the join pattern between the relations is arbitrary (theta-join) then no
- Equi-joins have a very regular pattern which can be exploited

Total time/space = #Nodes + #Edges = $O(n^2\ \ell)$

Factorized Representation of Join Query Results

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



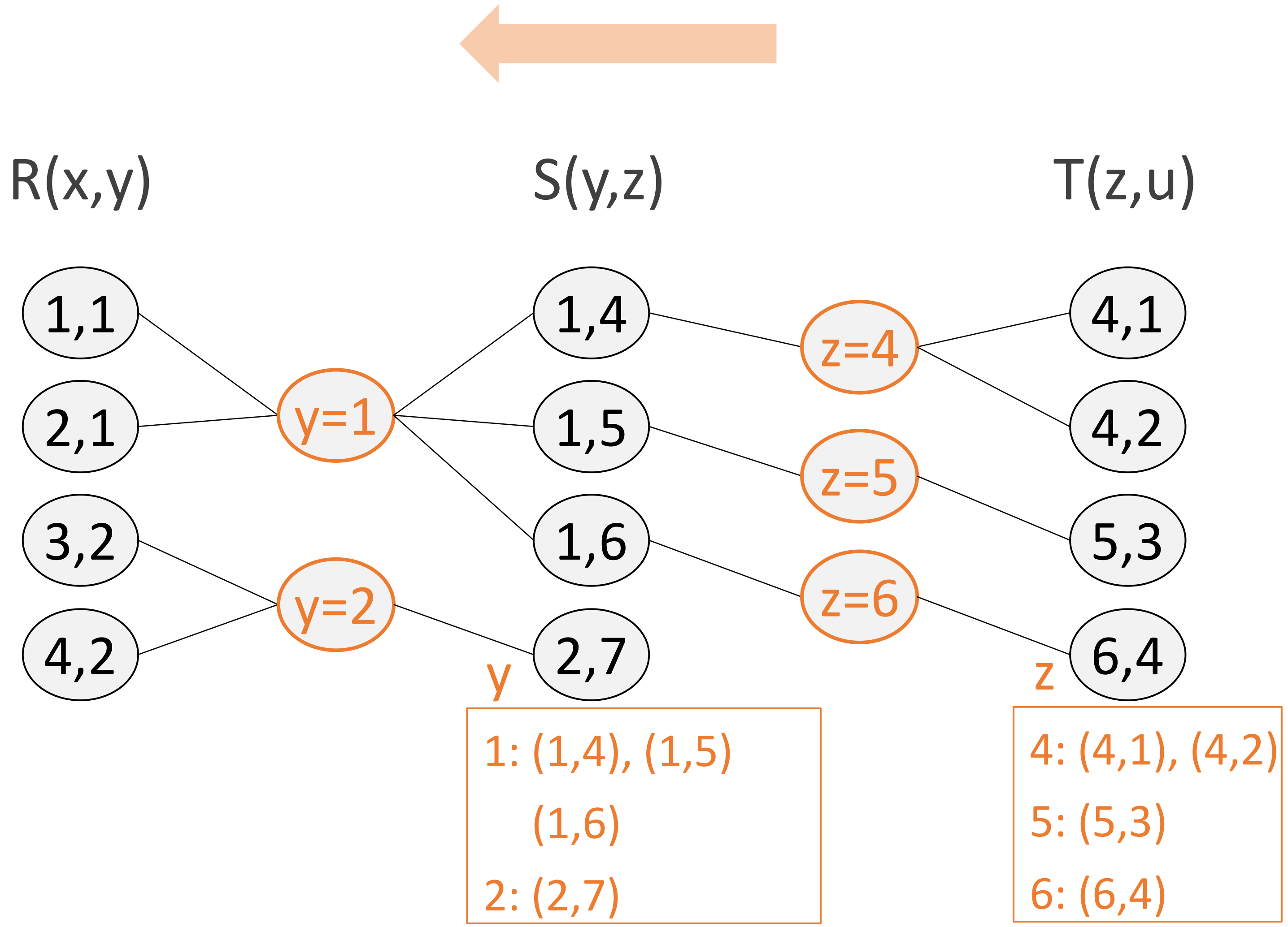
Further factorization:
Nodes in the middle create groups of common join values

Total time/space = #Nodes + #Edges = $O(n\,\ell)$

Linear in the size of the database

Factorized Representation Construction

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

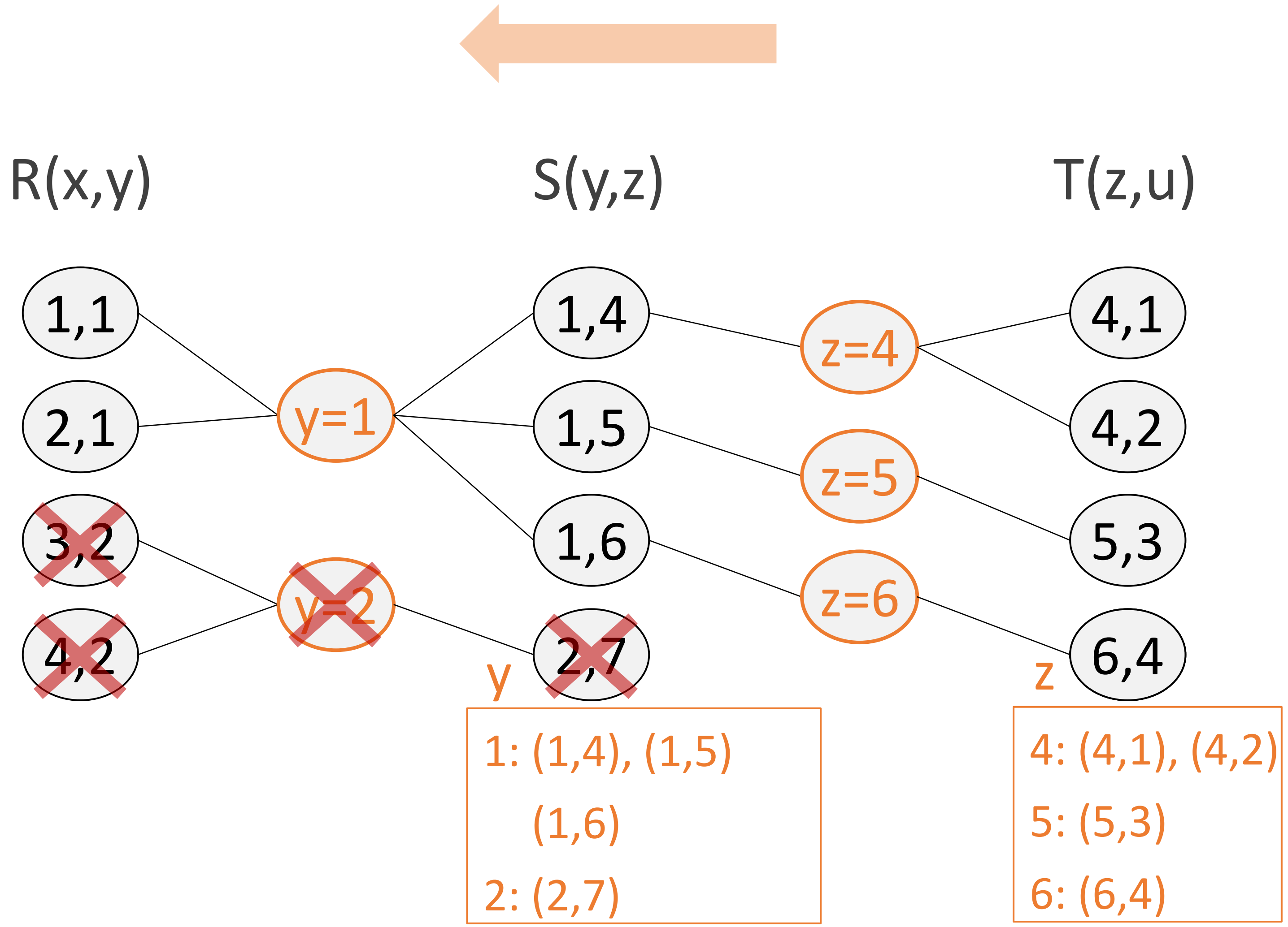


- How do we construct this representation?
- Bottom-up (right-to-left), using appropriate indexes on the relations

Hash Indexes

Semi-join Reduction

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



- Whenever a node on the left doesn't join with a node on the right, we can remove it
- Equivalent to the semi-join reduction of Yannakis
- Afterwards, no dead-ends if we traverse the representation top-down (left-to-right)

Hash Indexes

Outline Part 4

Part 4: Factorization

- High-level idea
- Factorized representation of path-CQ
- Factorized representation of tree-CQ & enumeration
- Tuple-level vs Attribute-level representations

Factorized Representation for Tree Query

R(x,y)

x	y
a_1	b_1
a_1	b_2
a_4	b_6

S(z,v)

z	v
c_1	d_1
c_1	d_2
c_4	d_6

T(p,x,y)

p	x	y
e_1	a_1	b_1
e_1	a_1	b_2
e_3	a_3	b_1
e_3	a_1	b_4
e_2	a_2	b_3

U(y)

y
b_1
b_2
b_3

W(u,x,y)

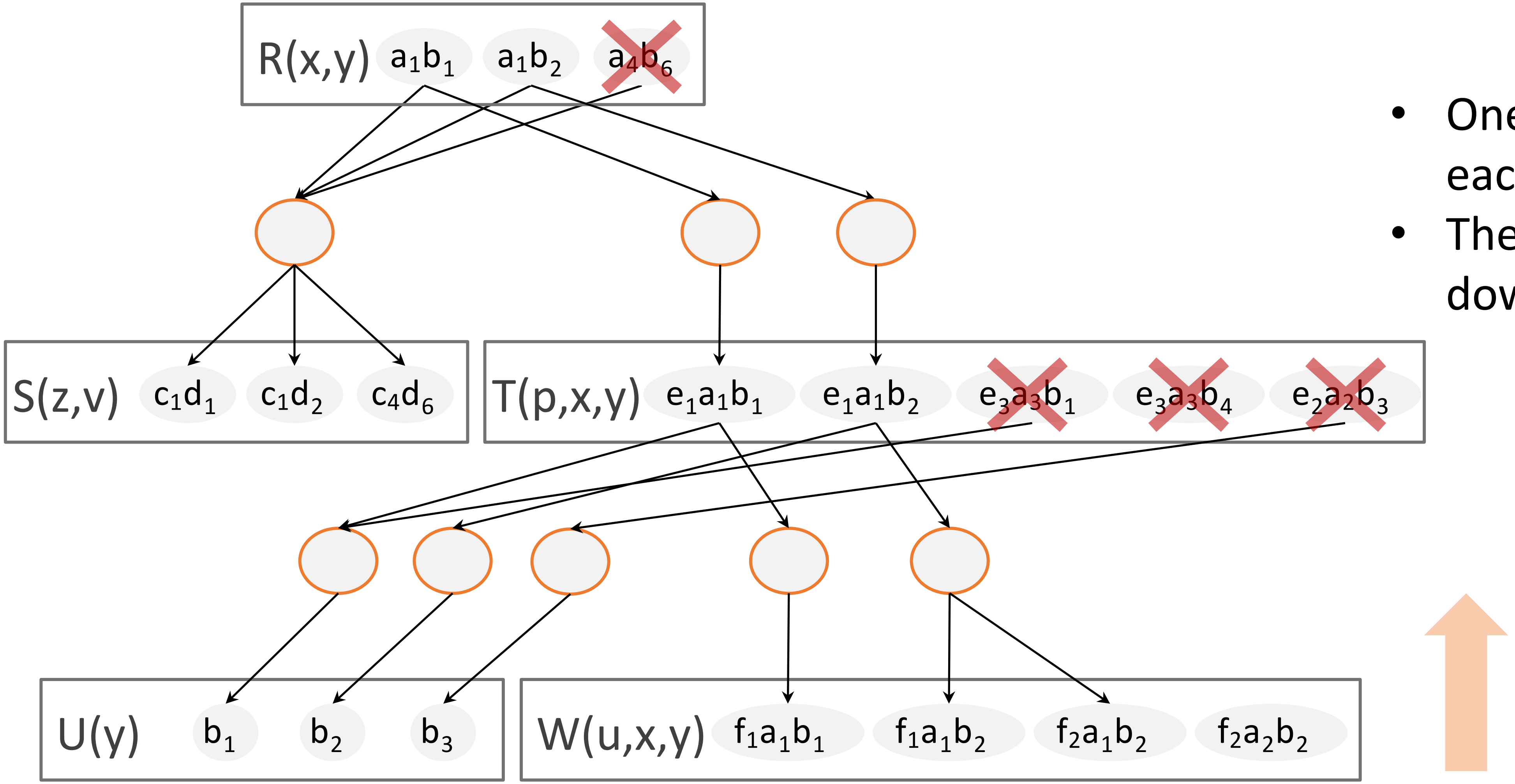
u	x	y
f_1	a_1	b_1
f_1	a_1	b_2
f_2	a_1	b_2
f_2	a_2	b_2

Q(x,y,z,v,p,u) :- R(x,y), S(z,v), T(p,x,y), U(y), W(u,x,y).

- In general, we construct the representation according to the join-tree order

Factorized Representation for Tree Query

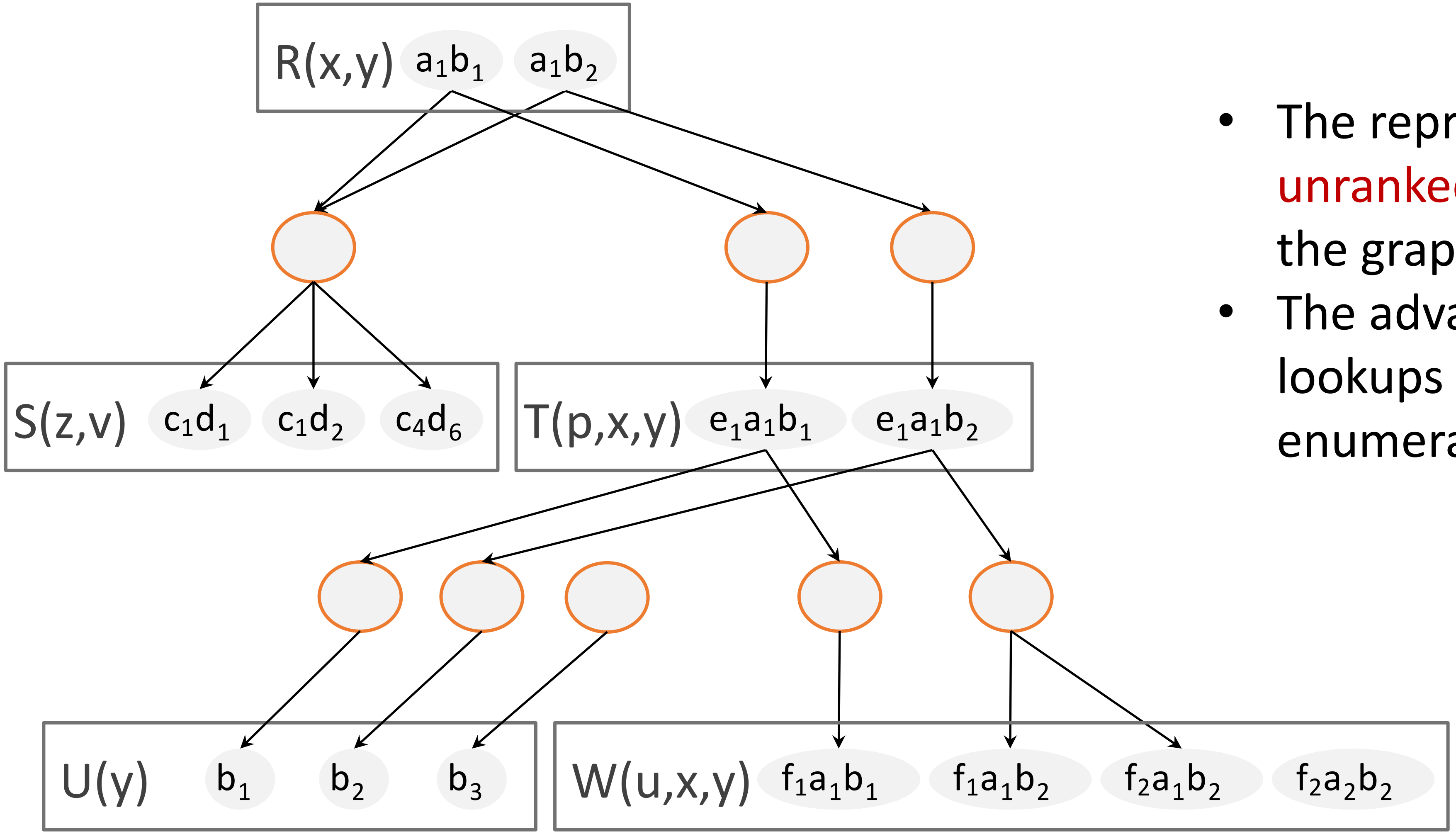
$Q(x,y,z,v,p,u) \text{ :- } R(x,y), S(z,v), T(p,x,y), U(y), W(u,x,y).$



- One layer of nodes for each relation
- The edges have a top-down direction

Factorized Representation for Tree Query

Q(x,y,z,v,p,u) :- R(x,y), S(z,v), T(p,x,y), U(y), W(u,x,y).



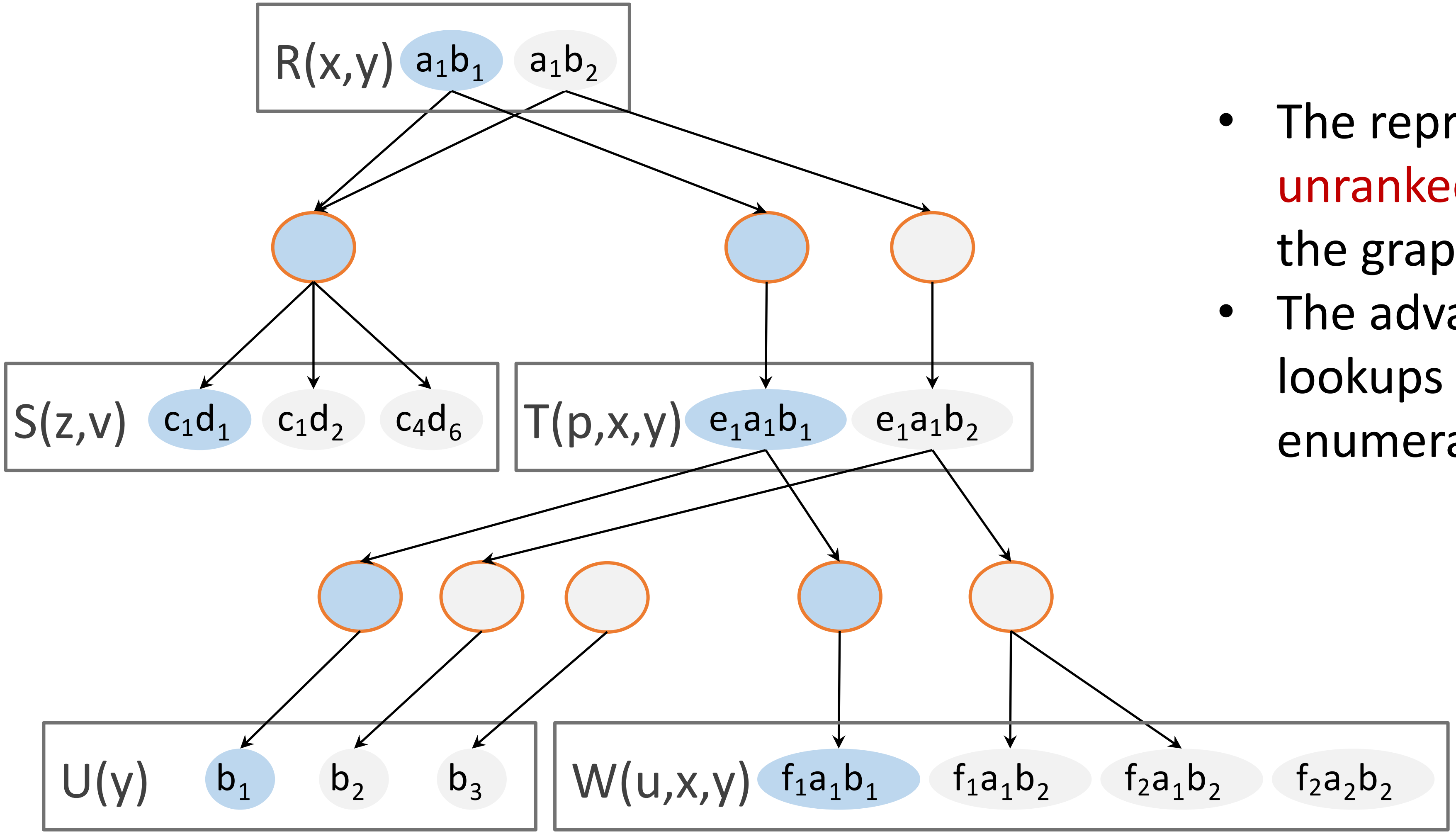
- The representation supports unranked enumeration by traversing the graph top-down
- The advantage now is that no hash lookups are required during enumeration

Join results

x	y	z	v	p	u

Factorized Representation for Tree Query

$Q(x,y,z,v,p,u) \text{ :- } R(x,y), S(z,v), T(p,x,y), U(y), W(u,x,y).$



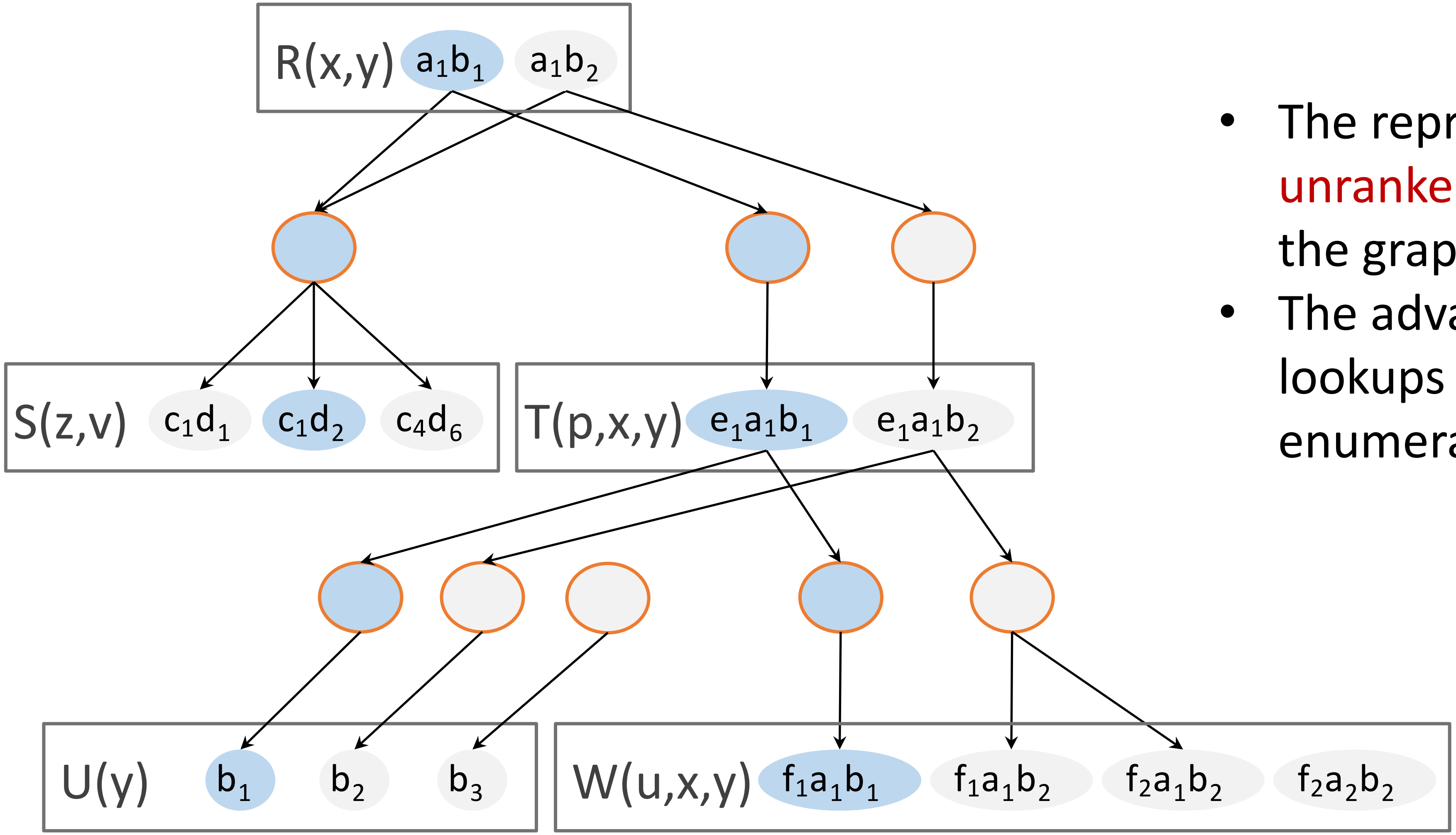
- The representation supports unranked enumeration by traversing the graph top-down
- The advantage now is that no hash lookups are required during enumeration

Join results

x	y	z	v	p	u
a_1	b_1	c_1	d_1	e_1	f_1

Factorized Representation for Tree Query

$Q(x,y,z,v,p,u)$:- $R(x,y)$, $S(z,v)$, $T(p,x,y)$, $U(y)$, $W(u,x,y)$.



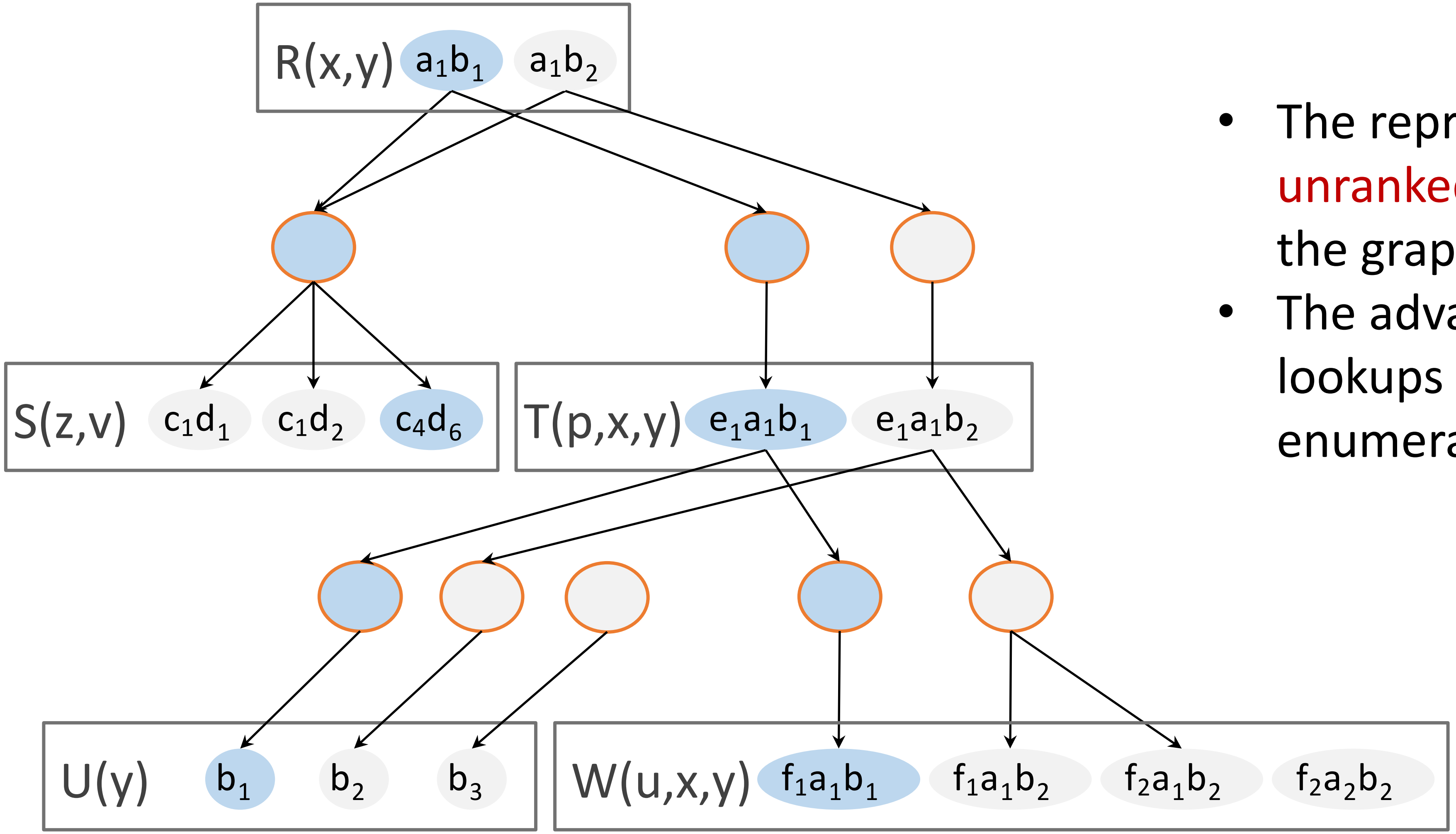
- The representation supports unranked enumeration by traversing the graph top-down
- The advantage now is that no hash lookups are required during enumeration

Join results

x	y	z	v	p	u
a_1	b_1	c_1	d_1	e_1	f_1
a_1	b_1	c_1	d_2	e_1	f_1

Factorized Representation for Tree Query

Q(x,y,z,v,p,u) :- R(x,y), S(z,v), T(p,x,y), U(y), W(u,x,y).



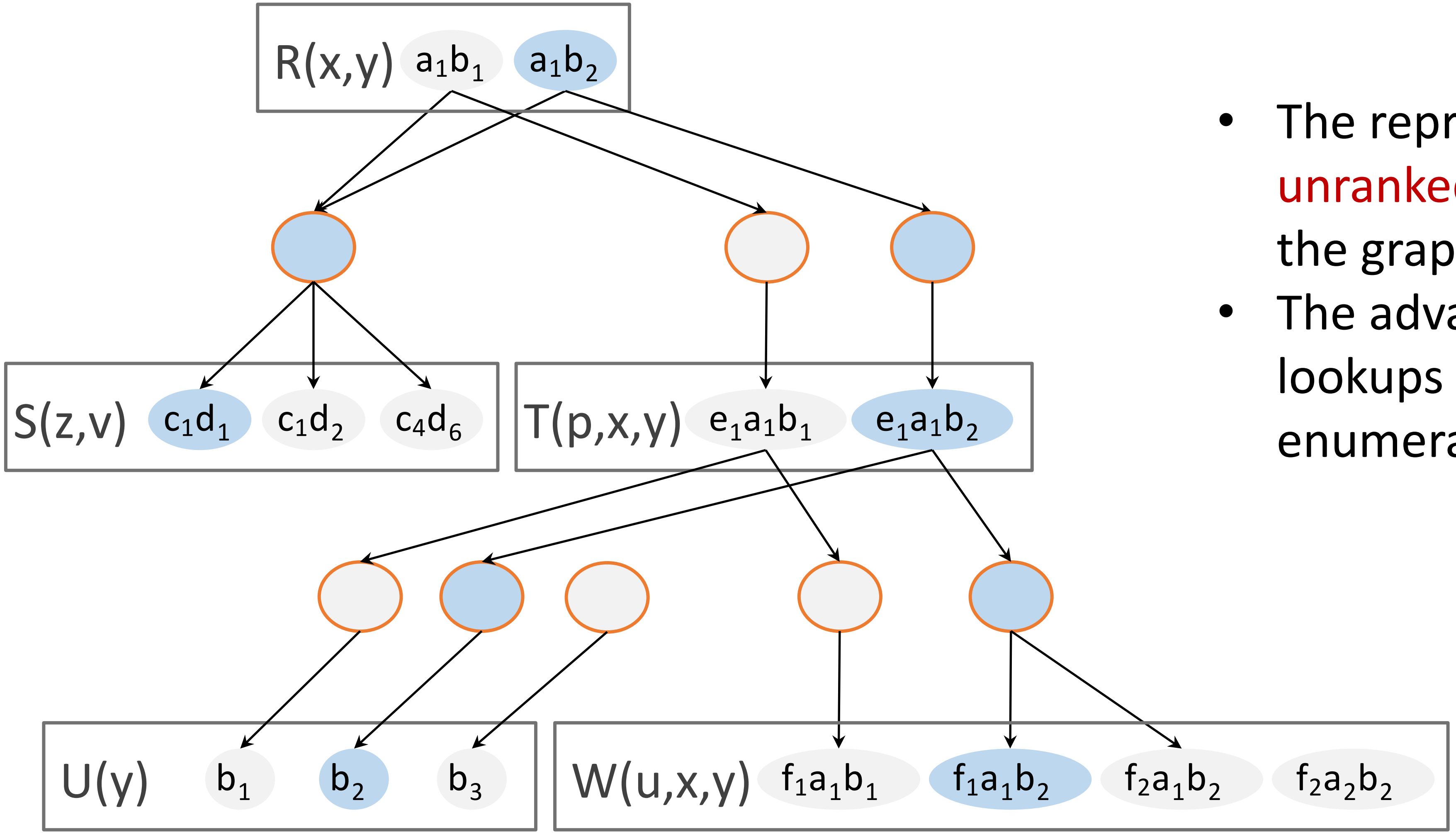
- The representation supports unranked enumeration by traversing the graph top-down
- The advantage now is that no hash lookups are required during enumeration

Join results

x	y	z	v	p	u
a_1	b_1	c_1	d_1	e_1	f_1
a_1	b_1	c_1	d_2	e_1	f_1
a_1	b_1	c_4	d_6	e_1	f_1

Factorized Representation for Tree Query

$Q(x,y,z,v,p,u)$:- $R(x,y)$, $S(z,v)$, $T(p,x,y)$, $U(y)$, $W(u,x,y)$.



- The representation supports unranked enumeration by traversing the graph top-down
- The advantage now is that no hash lookups are required during enumeration

Join results

x	y	z	v	p	u
a_1	b_1	c_1	d_1	e_1	f_1
a_1	b_1	c_1	d_2	e_1	f_1
a_1	b_1	c_4	d_6	e_1	f_1
a_1	b_2	c_1	d_1	e_1	f_1
…					

Outline Part 4

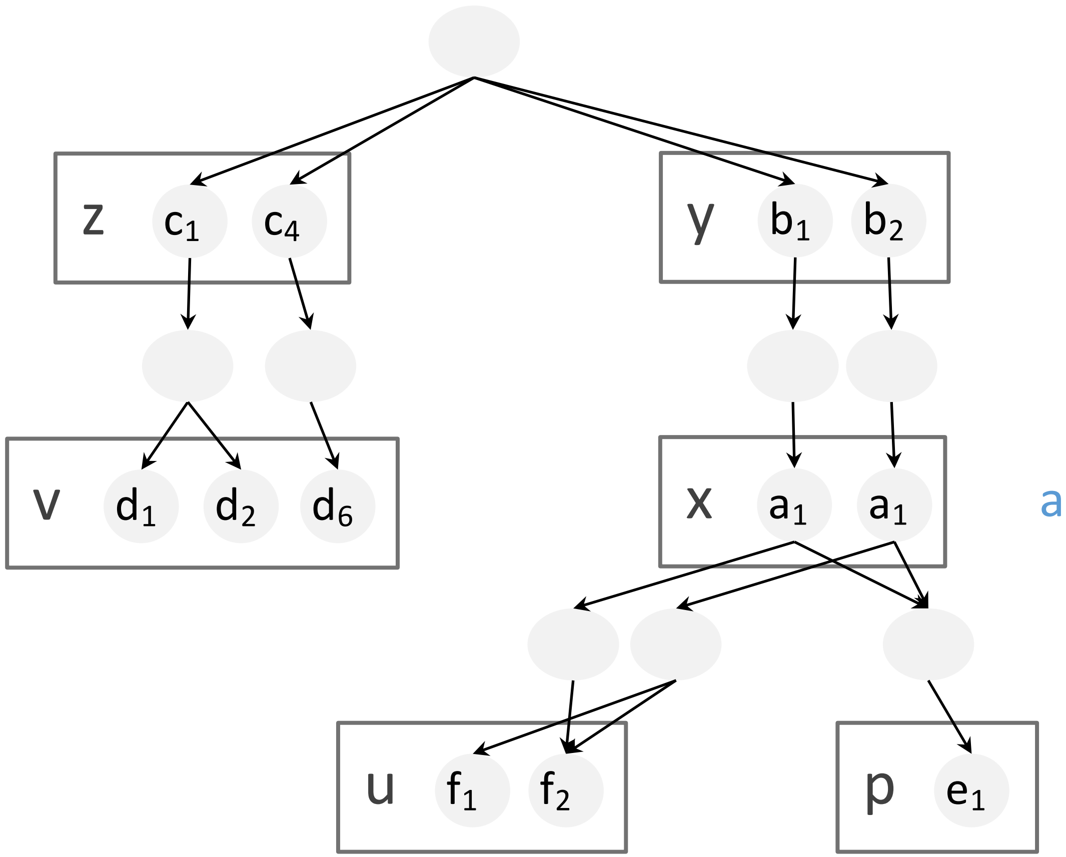
Part 4: Factorization

- High-level idea
- Factorized representation of path-CQ
- Factorized representation of tree-CQ & enumeration
- Tuple-level vs Attribute-level representations

Attribute-level vs Tuple-level Factorizations

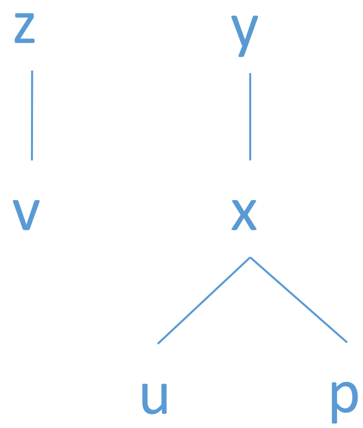
- Dual perspective: nodes are attribute values instead of tuples
- Formalized by work on factorized databases

(The actual representation used by factorized databases is in the form of circuit with union and product nodes that is equivalent for join queries)



a_1 repeated

"d-tree" instead of join tree
(actually, a forest here)



Dependent attributes need to be on the same root-to-leaf path

Olteanu, Závodný. Size bounds for factorised representations of query results. TODS 2015 https://doi.org/10.1145/2656335
Towards Responsive DBMS. ICDE 2022 tutorial: https://northeastern-datalab.github.io/responsive-dbms-tutorial

Attribute-level vs Tuple-level Factorizations

- Key differences for attribute-level:
  - The structure is not given by the join tree, but instead by a "d-tree"
  - Nodes corresponding to the same value can be repeated
  - Also factorizes the individual relations, which is not possible if a tuple is one unit
  - Theory on how to factorize query results with cycles or projections directly, without tree-decompositions or free-connex transformation
- Both can be constructed in $O(n)$ for full acyclic CQs (without projections)
- A lot of work beyond unranked enumeration
  - Enumeration with lexicographic orders
  - Learning models directly on the factorized representation
  - Maintenance under updates

Elghandour, Kara, Olteanu, Vansummeren. Incremental Techniques for Large-Scale Dynamic Query Processing. CIKM'18. https://doi.org/10.1145/3269206.3274271
Bakibayev, Kocisky, Olteanu, Zavodny. Aggregation and ordering in factorised databases. PVLDB'13. https://doi.org/10.14778/2556549.2556579
Schleich, Olteanu, Ciucanu. Learning linear regression models over factorized joins. SIGMOD'16. https://doi.org/10.1145/2882903.2882939
Kara, Ngo, Nikolic, Olteanu, Zhang. Maintaining triangle queries under updates. TODS 2020. https://doi.org/10.1145/3396375