Hypertree Decompositions Q&A

Georg Gottlob

University of Oxford

Joint work with

Gianluigi Greco, Nicola Leone, and Francesco Scarcello

What is this all about?

Hypertree Decompositions (HDs): a hypergraph-based method for automatically decomposing conjunctive queries and generating efficient query plans.

Each query is associated with its hypertree-width.

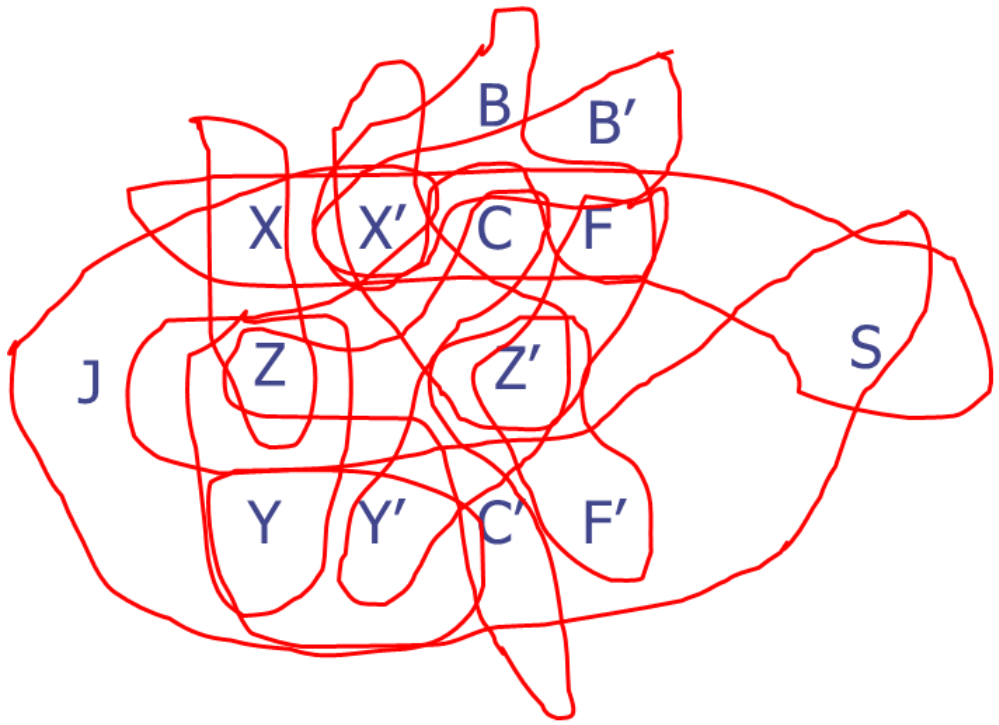
Queries of bounded hypertree width generalize *acyclic* queries and enjoy similar good properties.

HDs at a glance:

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$

HDs at a glance:

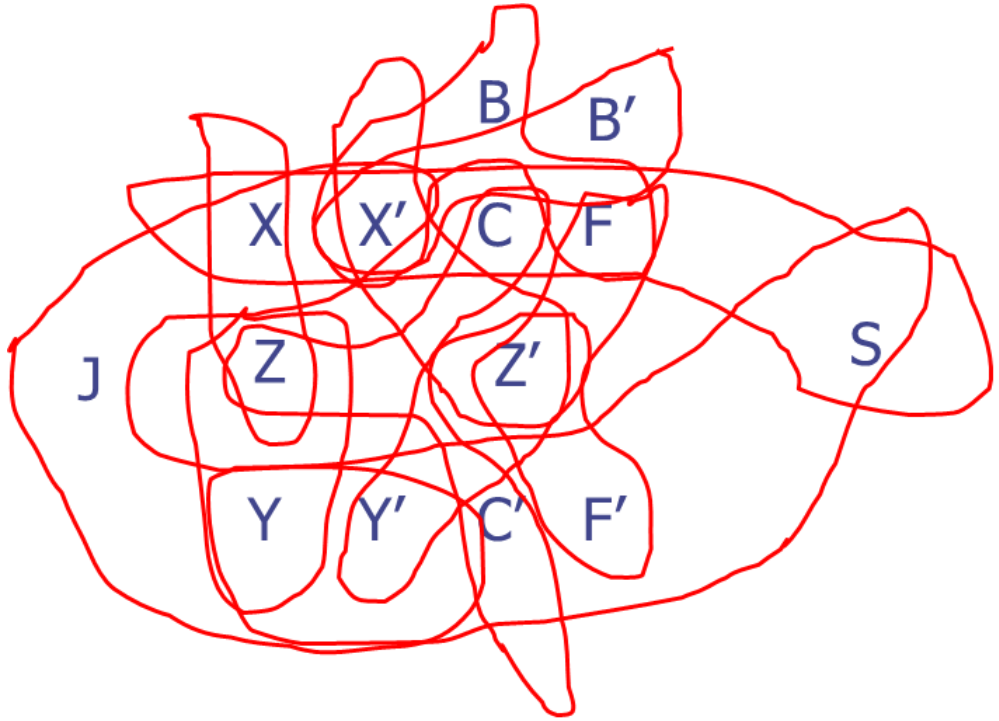
$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$



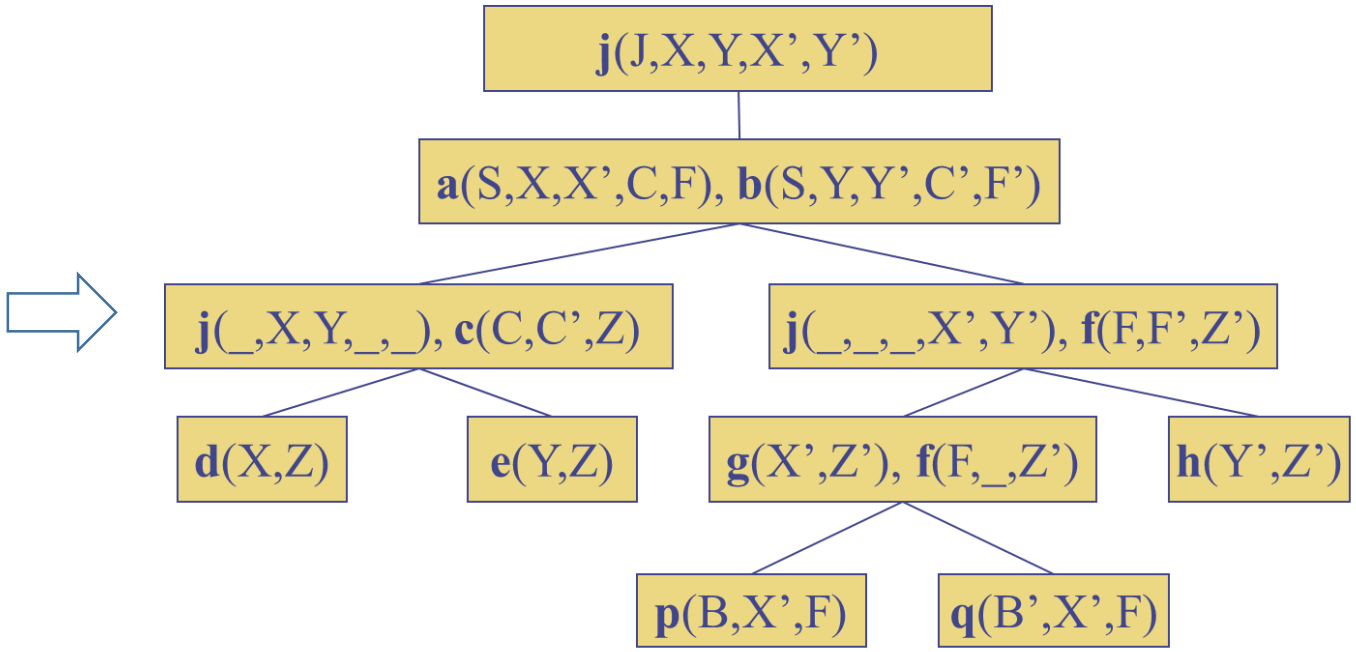
query hypergraph

HDs at a glance:

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$



query hypergraph



hypertree decomposition (query plan)

What is so smart about acyclic queries?

Acyclic conjunctive query (ACQ):

A Query whose associated hypergraph is acyclic (more precisely,α-acyclic [Fagin 83])

Query acyclicity was independently defined by

- [Beeri et al. STOC81] *acyclic database schemas*, and
- [Goodman & Shmueli 1981, TODS'82] *tree queries*

[Graham; Yu & Özsoyoğlu] *GYO reduction*

A query is acyclic iff it has a join tree (see next slide)

Acyclic conjunctive query (ACQ):

A Query whose associated hypergraph is acyclic (more precisely, α-acyclic **[Fagin 83]**)

Query acyclicity was independently defined by

- [Beeri et al. STOC81] *acyclic database schemas*, and
- [Goodman & Shmueli 1981, TODS'82] *tree queries*

[Graham; Yu & Özsoyoğlu] *GYO reduction*

A query is acyclic iff it has a join tree (see next slide)

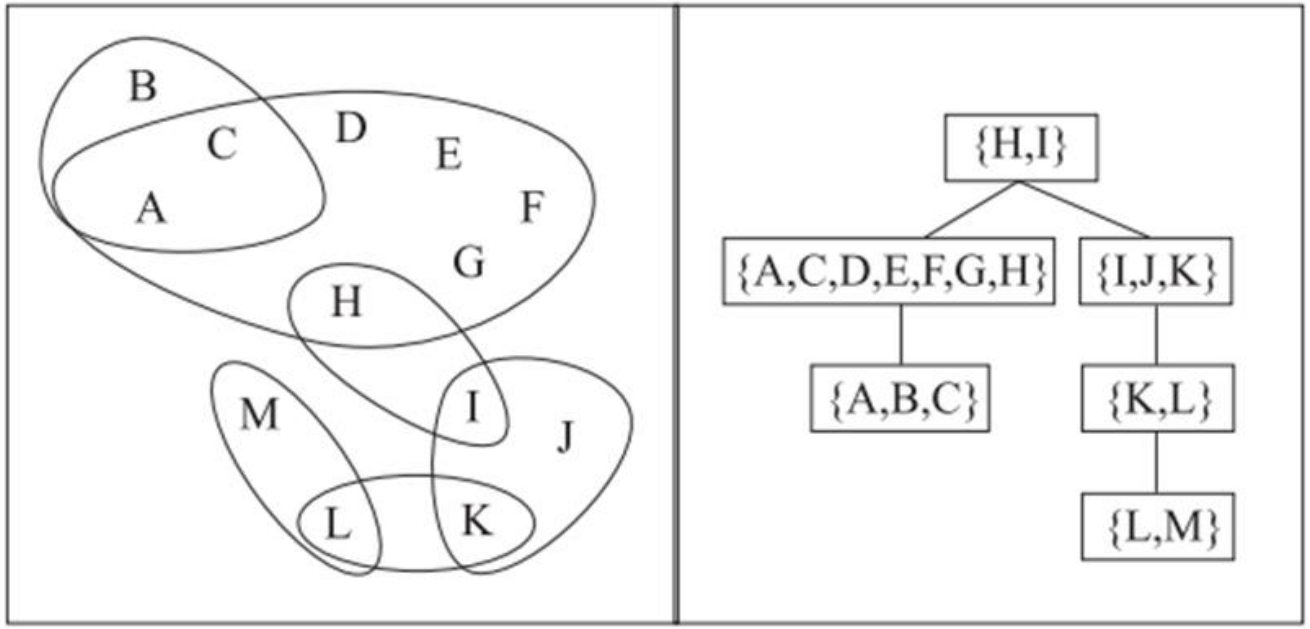
Good properties:

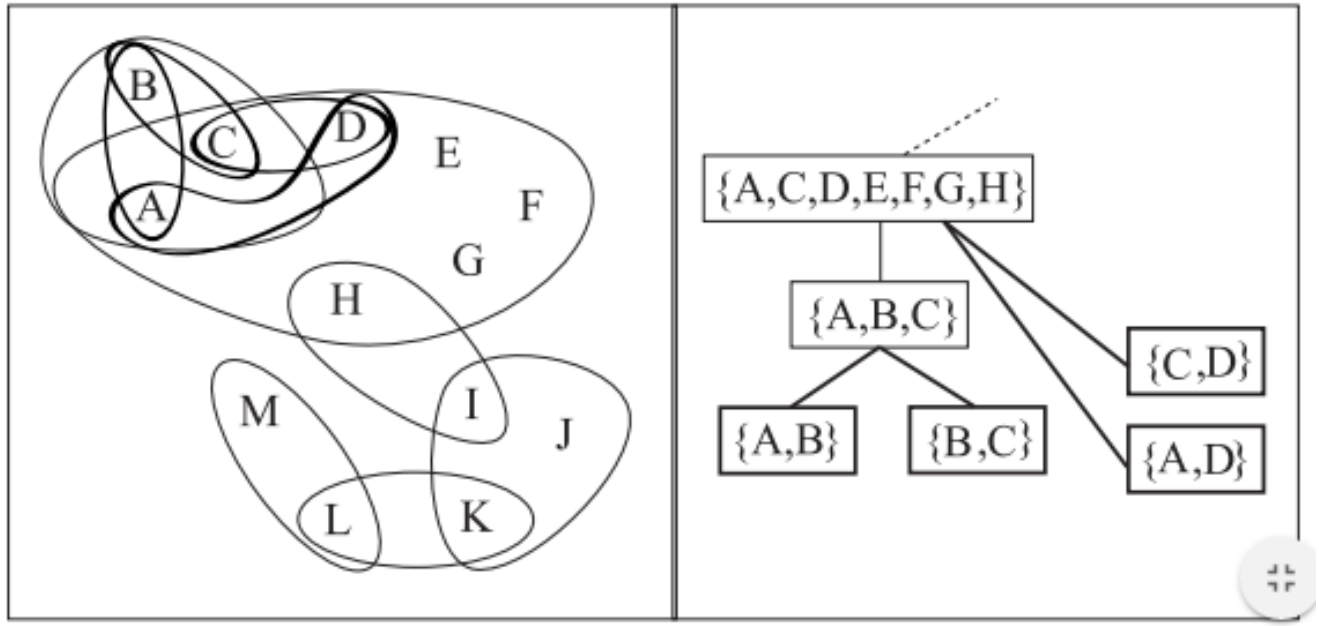
- ACQs can be recognized in polynomial (actually linear) time,
- A join-tree for an ACQ can be built in linear time,
- A Bolean ACQ Q can be answered in time $O(|Q| \times |r_{max}| \times \log |r_{max}|)$ [Yan'81]
- A non-Boolean ACQ can be answered with polynomial delay.

Acyclic queries & join trees

Join tree: a tree whose nodes are labelled by query hyperedges (or query atoms) such that:

- each hyperedge labels some node, and
- For each query variable V, the tree-nodes containing V span a connected subtree (*connectednes condition*)

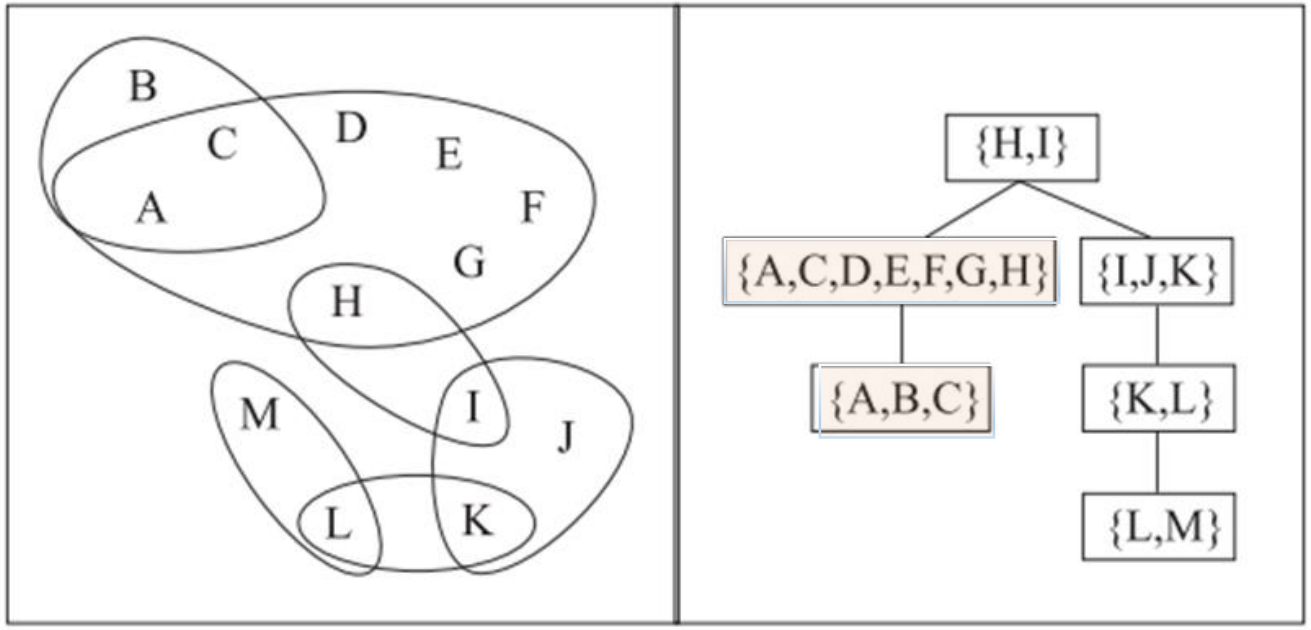


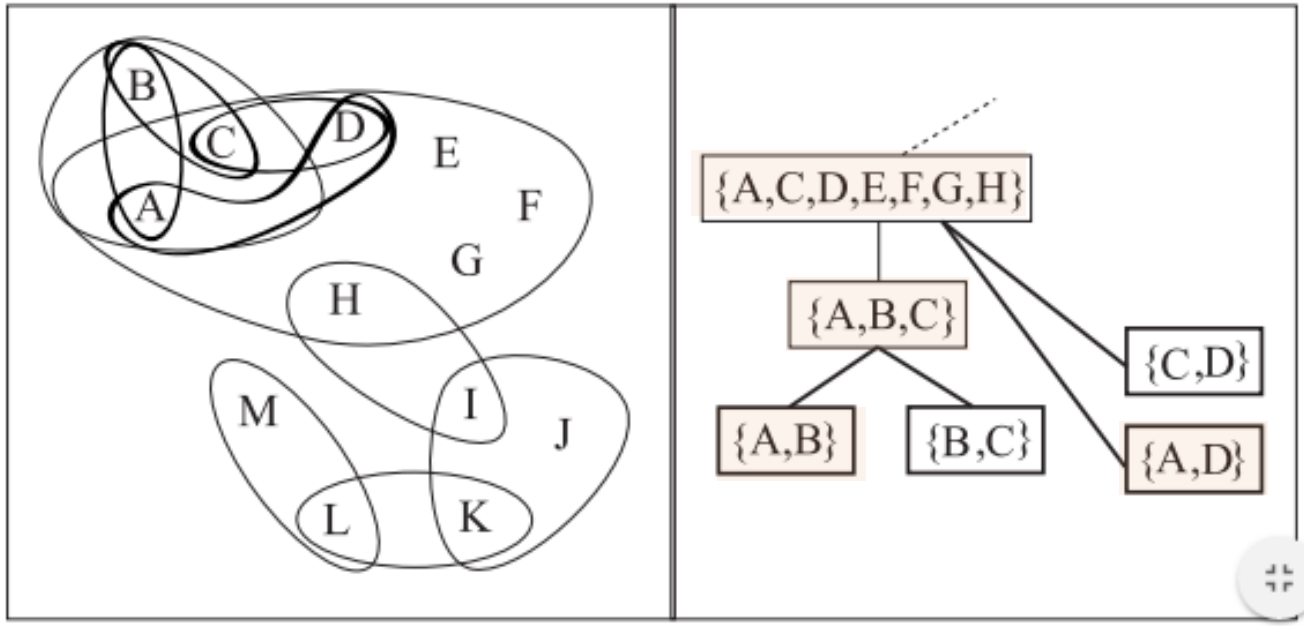


Acyclic queries & join trees

Join tree: a tree whose nodes are labelled by query hyperedges (or query atoms) such that:

- each hyperedge labels some node, and
- For each query variable V, the tree-nodes containing V span a connected subtree (*connectednes condition*)





Cool: Acyclic queries may contain cyclic sub-queries.

An ACQ may be easier to answer than its sub-queries.

Extreme example:

Q_n: $R(X_1,X_2,\ldots,X_n)$ & $E(X_1,X_2)$ & $E(X_1,X_3)$ &…& $E(X_i,X_j)$ …&$E(X_{n-1},X_n)$

Cool: Acyclic queries may contain cyclic subqueries.

An ACQ may be easier to answer than its subqueries.

Extreme example:

Q_n: $R(X_1,X_2,\ldots,X_n)$ & $E(X_1,X_2)$ & $E(X_1,X_3)$ &…& $E(X_i,X_j)$ …&$E(X_{n-1},X_n)$

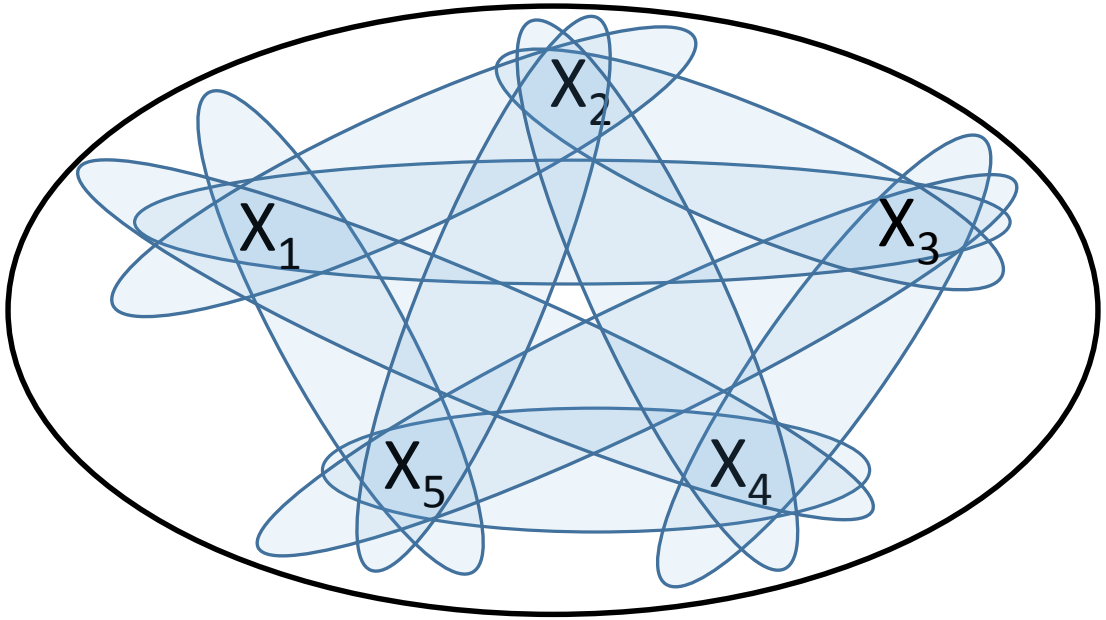
NP-complete clique-subquery

Cool: Acyclic queries may contain cyclic subqueries.

An ACQ may be easier to answer than its subqueries.

Extreme example:

Q_n: $R(X_1,X_2,\ldots,X_n)$ & $E(X_1,X_2)$ & $E(X_1,X_3)$ &…& $E(X_i,X_j)$ …&$E(X_{n-1},X_n)$

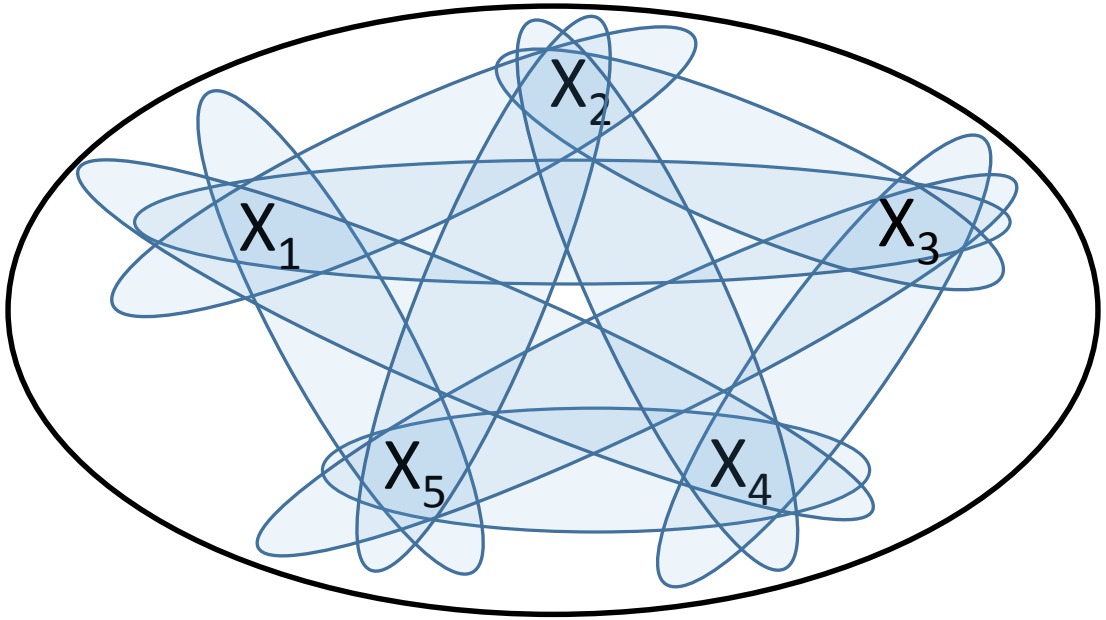


Cool: Acyclic queries may contain cyclic subqueries.

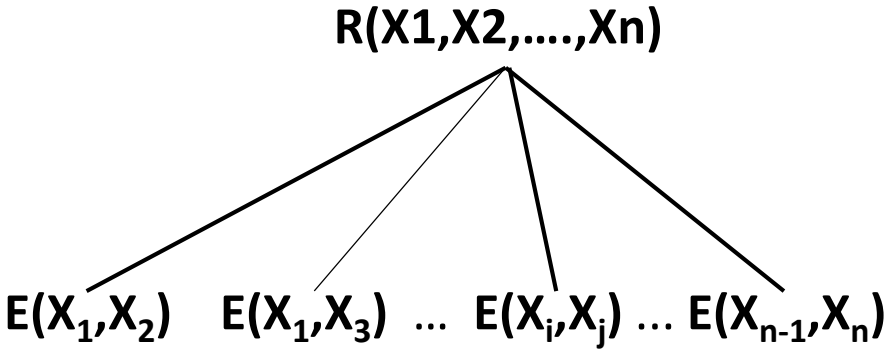
An ACQ may be easier to answer than its subqueries.

Extreme example:

Q_n: $R(X_1,X_2,\ldots,X_n)$ & $E(X_1,X_2)$ & $E(X_1,X_3)$ &…& $E(X_i,X_j)$ …&$E(X_{n-1},X_n)$



Join tree for Q_n:



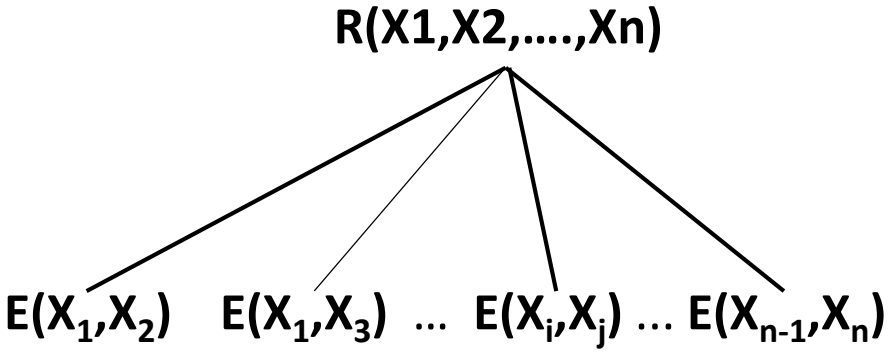
Cool: Acyclic queries may contain cyclic subqueries.

An ACQ may be easier to answer than its subqueries.

Extreme example:

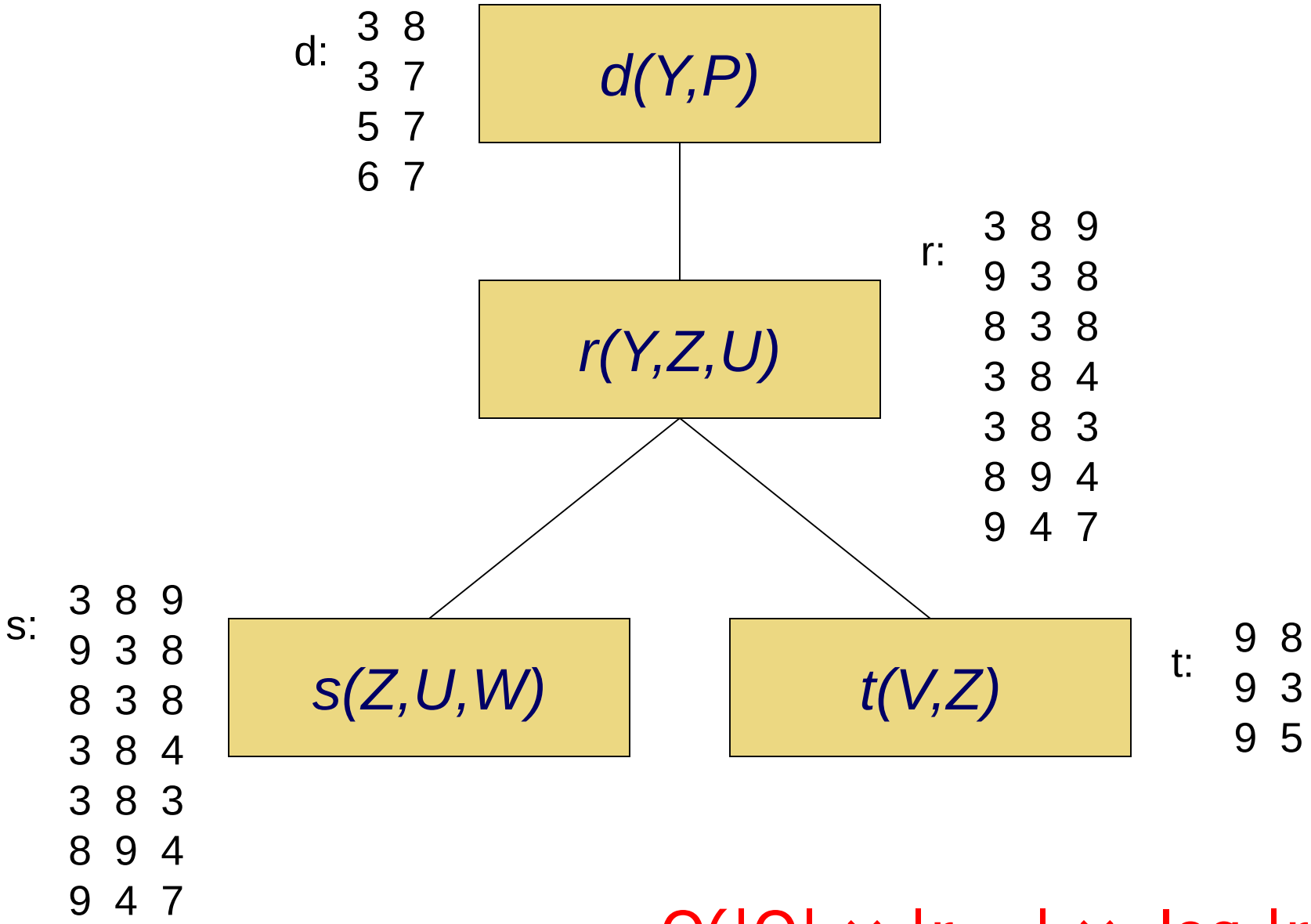
Q_n: $R(X_1,X_2,\ldots,X_n)$ & $E(X_1,X_2)$ & $E(X_1,X_3)$ &…& $E(X_i,X_j)$ …&$E(X_{n-1},X_n)$

Join tree for Q_n:

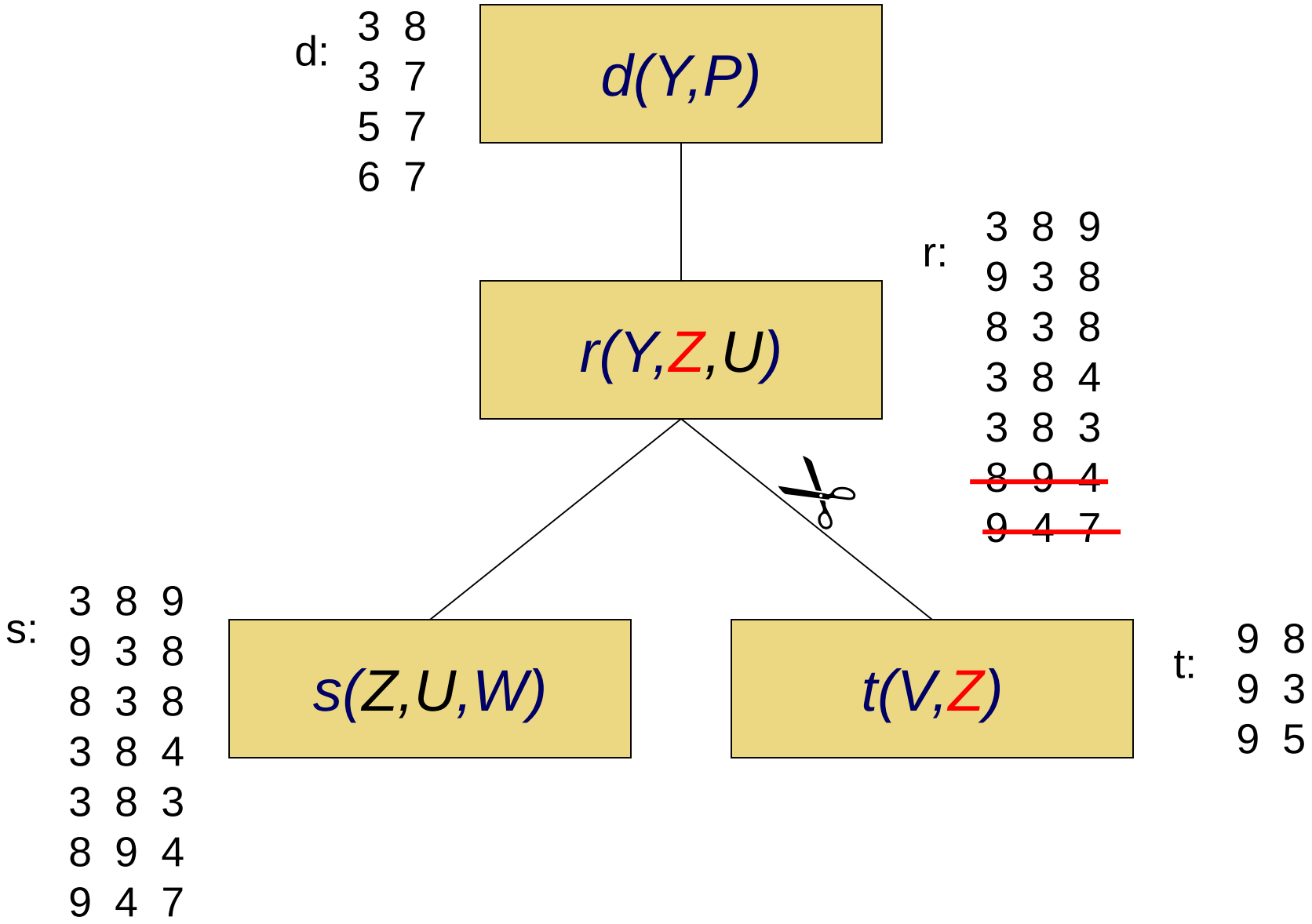


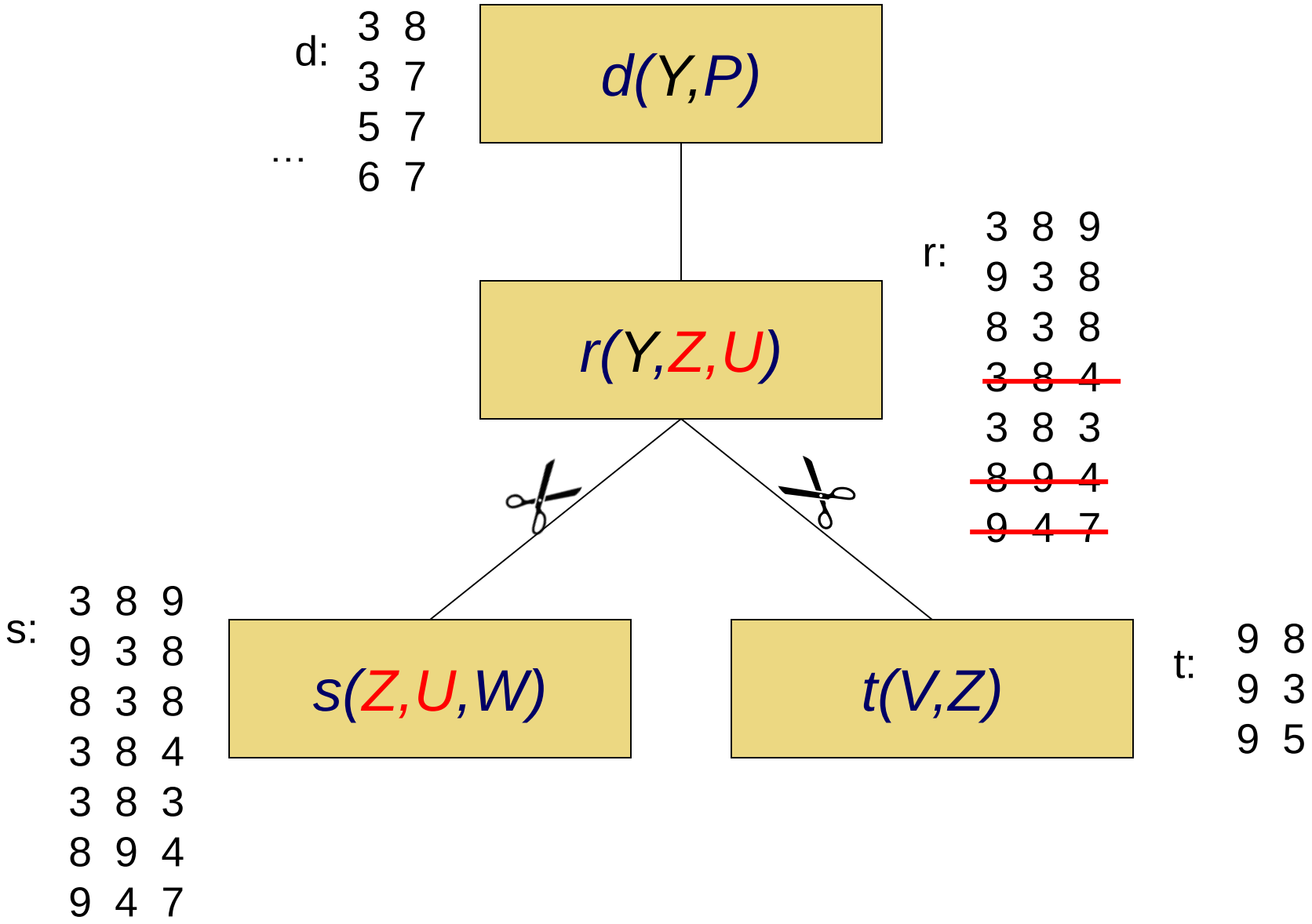
The order of evaluation matters!
This is automatically taken care of
by acyclic query evaluation algorithms
(query plans)

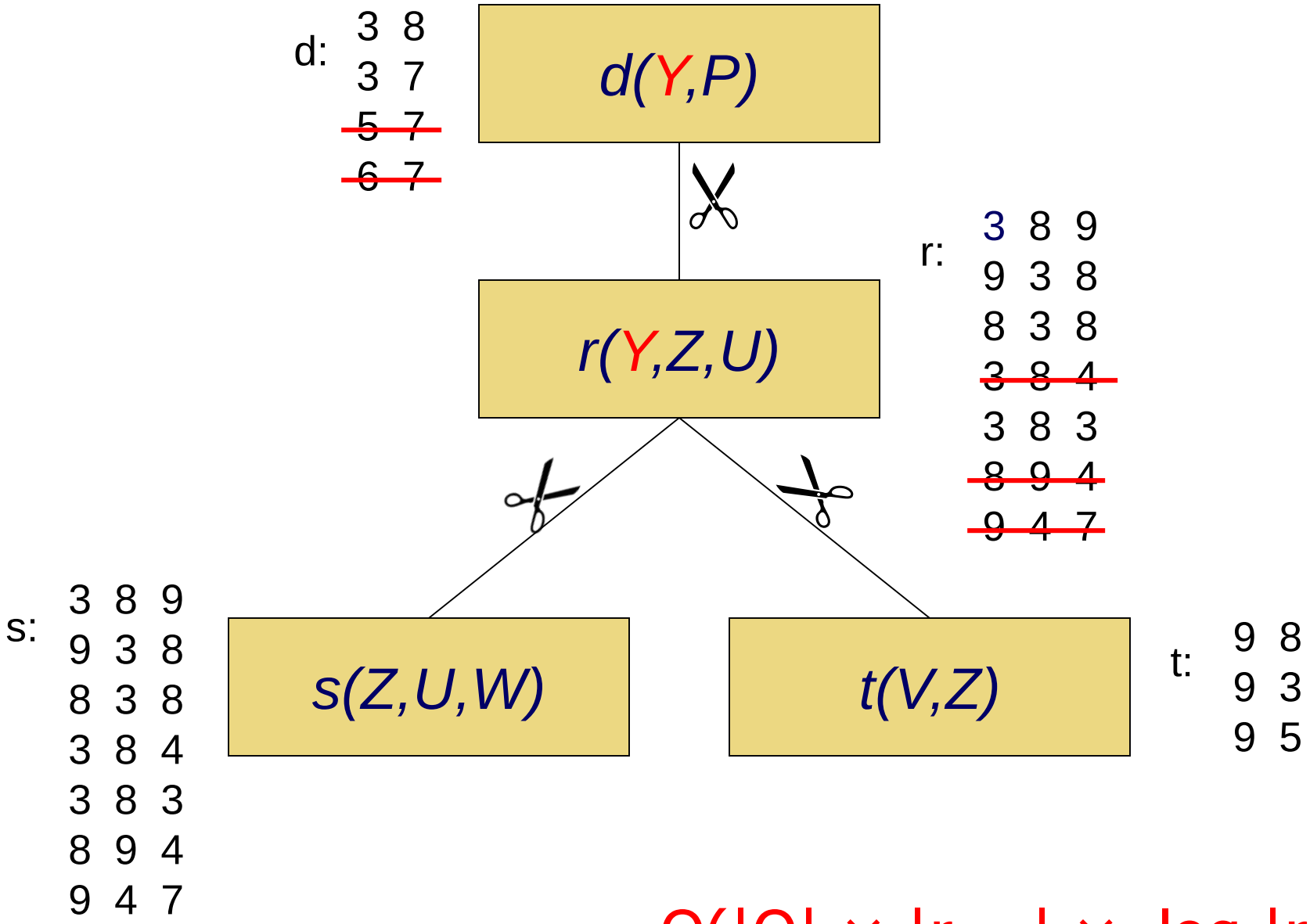
How are ACQs evaluated according to Yannakakis‘Algorithm?



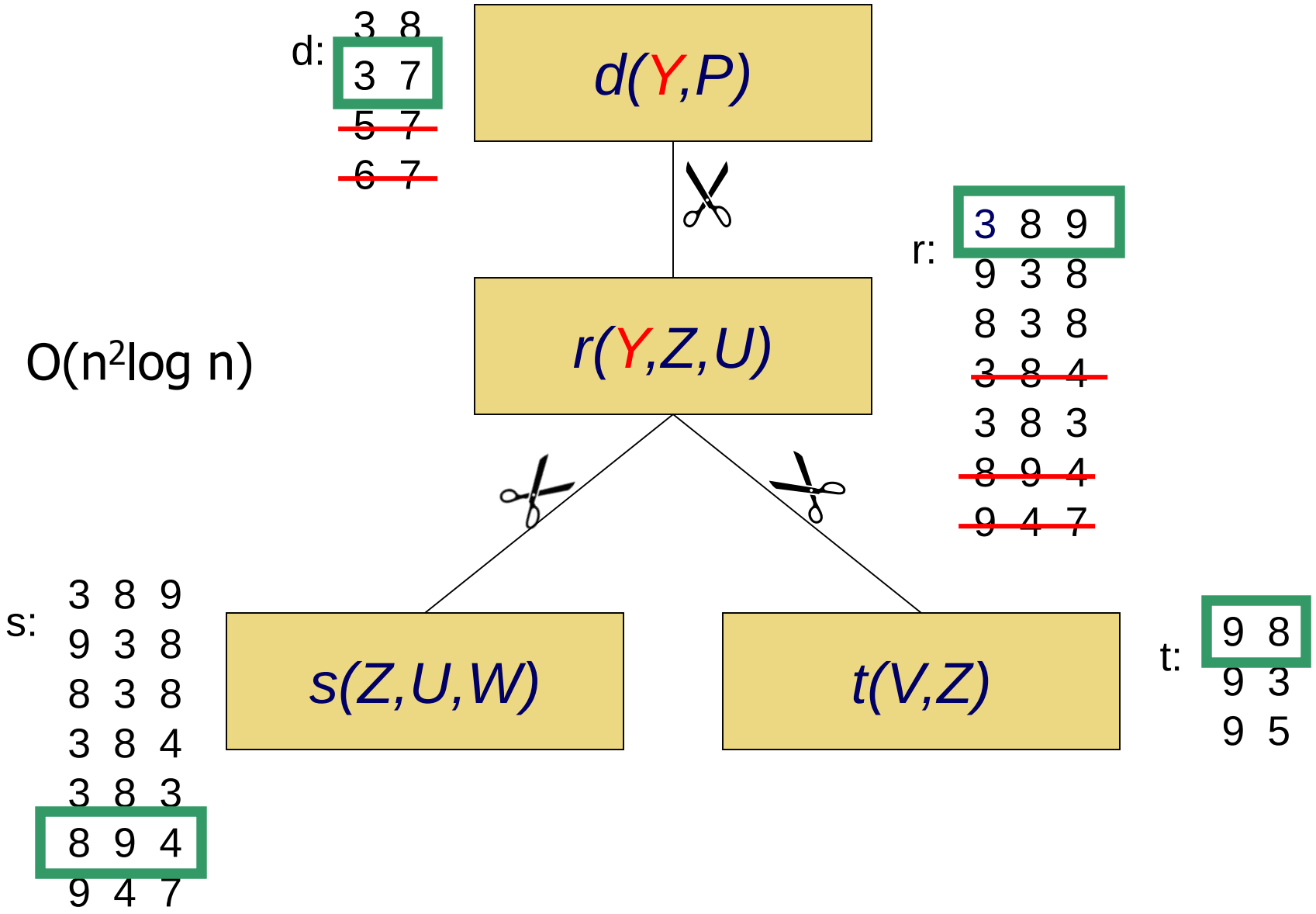
$$O(|Q| \times |r_{max}| \times \log |r_{max}|)$$


d:
3 8
3 7
5 7
6 7
d(Y,P)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
r(Y,Z,U)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
t(V,Z)
t:
9 8
9 3
9 5


d:
3 8
3 7
5 7
...
6 7
d(Y,P)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
r(Y,Z,U)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
t(V,Z)
t:
9 8
9 3
9 5



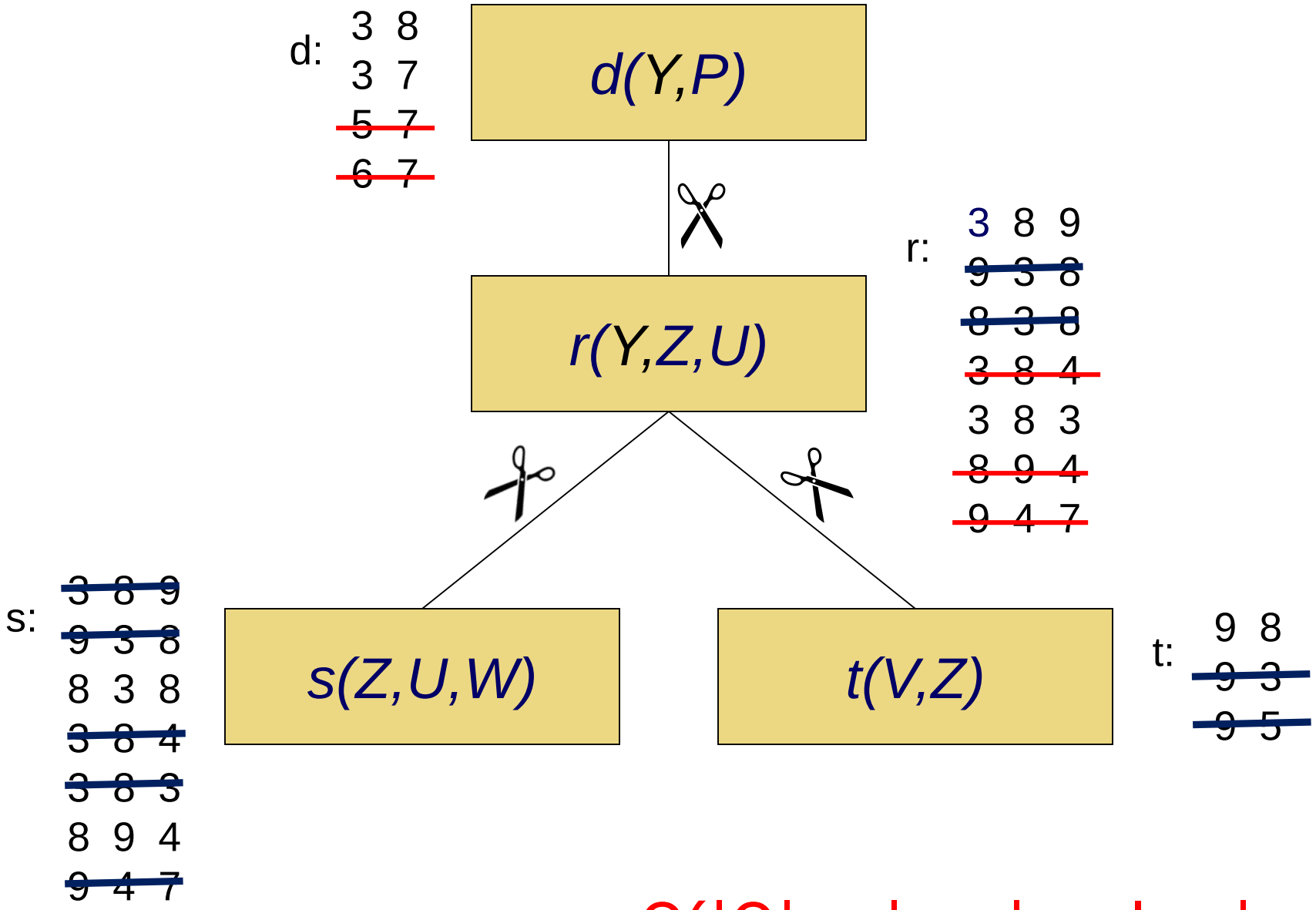
$$O(|Q| \times |r_{max}| \times \log |r_{max}|)$$



$O(n^2 \log n)$

A solution: Y=3, P=7, Z=8, U=9, W=4, V=9

To obtain fully reduced relations, perform semijoins downwards



$O(|Q| \times |r_{max}| \times \log |r_{max}|)$

How to generalize query acyclicity?

Generalizations come with a **width** expressing the degree of cyclicity

Good generalizations should fulfill 3 criteria:

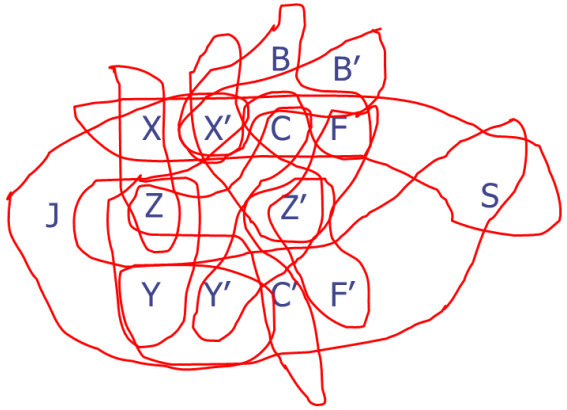
1. **Generalization of Acyclicity:** Queries of width k≥1 include all acyclic ones.
2. **Tractable Recognizability:** Width k queries can recognized efficiently.
3. **Tractable Query-Answering:** Width k queries can be answered efficiently.

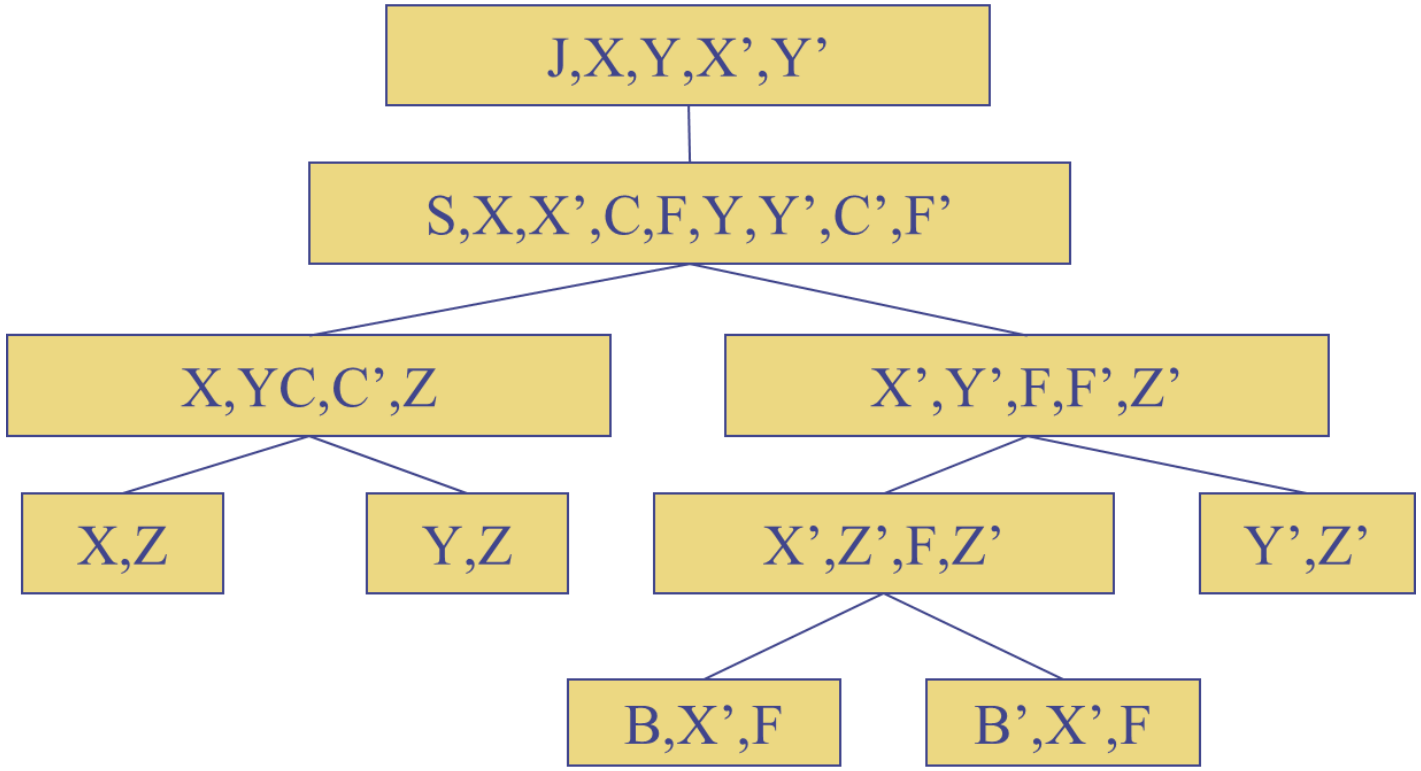
Main Previous Generalization Attempts

- **Treewidth:** violates Criterion 1 (generalization), see also next slide.
- **Hinge width:** violates Criterion 1 (= generalization).
- **Query width:** violates Criterion 2 (= tractable recognizability),
  moreover, leads to a suboptimal width.

Tree Decomposition

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$

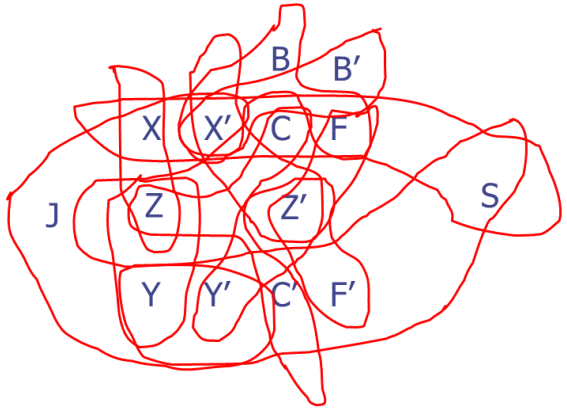


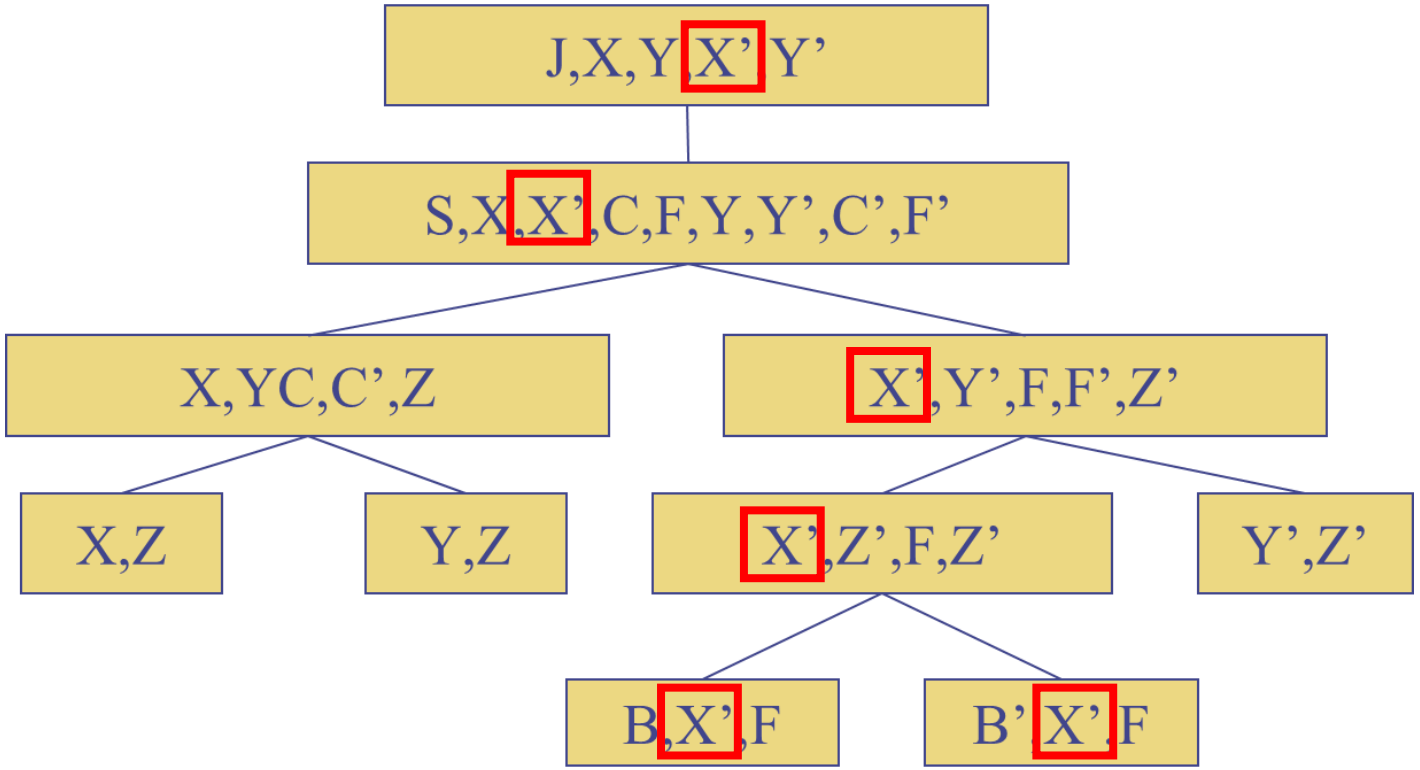


- Variables of each atom covered by some bag
- Connectedness condition

Tree Decomposition

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$

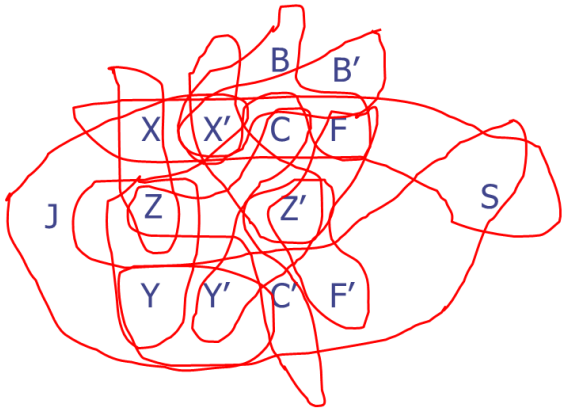


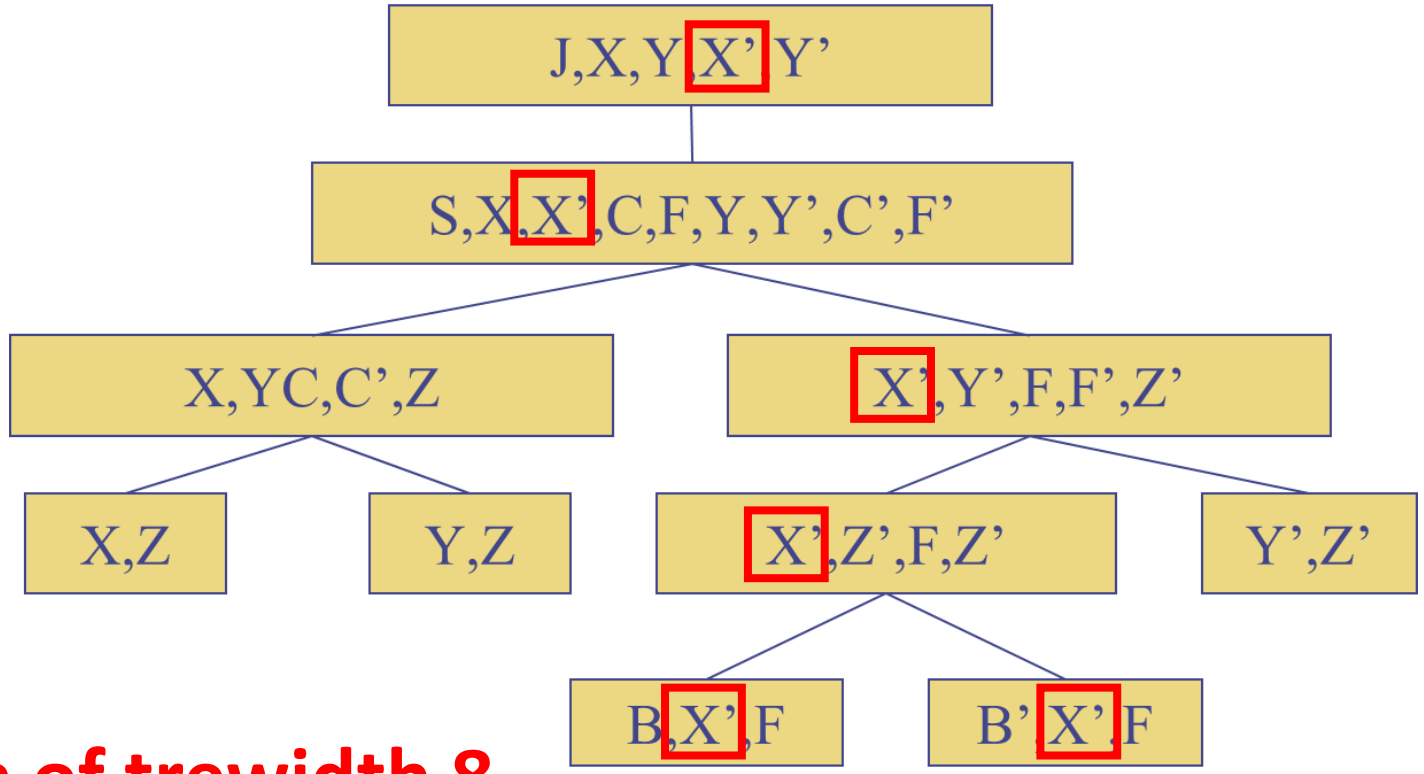


- Variables of each atom covered by some bag
- Connectedness condition

Tree Decomposition

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$





- Variables of each atom covered by some bag
- Connectedness condition

Decomposition of trewidth 8

The good:

- Optimal tree decomps of fixed width k can be computed efficiently.

  [Robertson-Seymour 86],[Bodlaender 93], [follow-up works]

- Queries of bounded treewidth k can be evaluated efficiently

  [Chekuri-Rajaraman 97], [Kolaitis-Vardi 98], [CSP-Literature]

  Straightforward bound implied by these works: at least $O(\ |dom|^{k+1} \times |Q|\ )$

  Different bound (based on HT-ideas): $O(|Q| \times |r_{max}|^{\lfloor k+1/2 \rfloor} \times \log |r_{max}|)$

The bad:

- Bounded treewidth does not generalize acyclicity

Example: $Q_n = R(X_1, X_2, \ldots, X_n)$;

$tw(Q_n) = n-1$

$X_1, X_2, \ldots, X_n$

tree decomposition

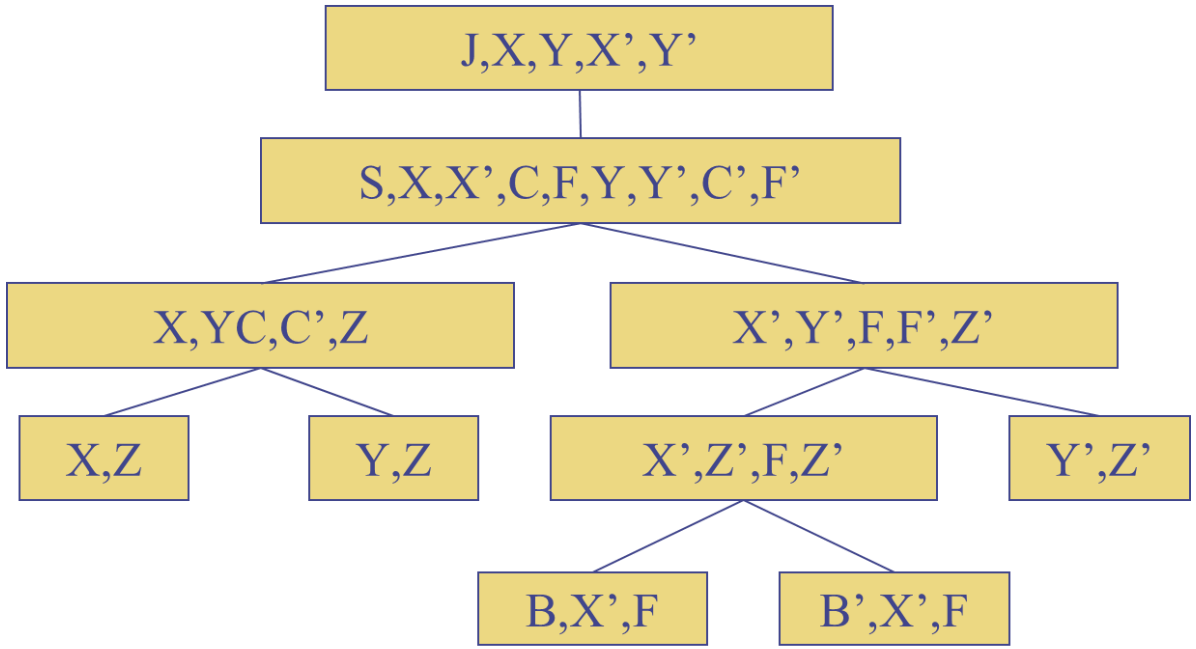
Tree decompositions and treewidth dramatically fail to capture the essence of query acyclicity.

This is the case even for classes of queries of fixed arity > 2.

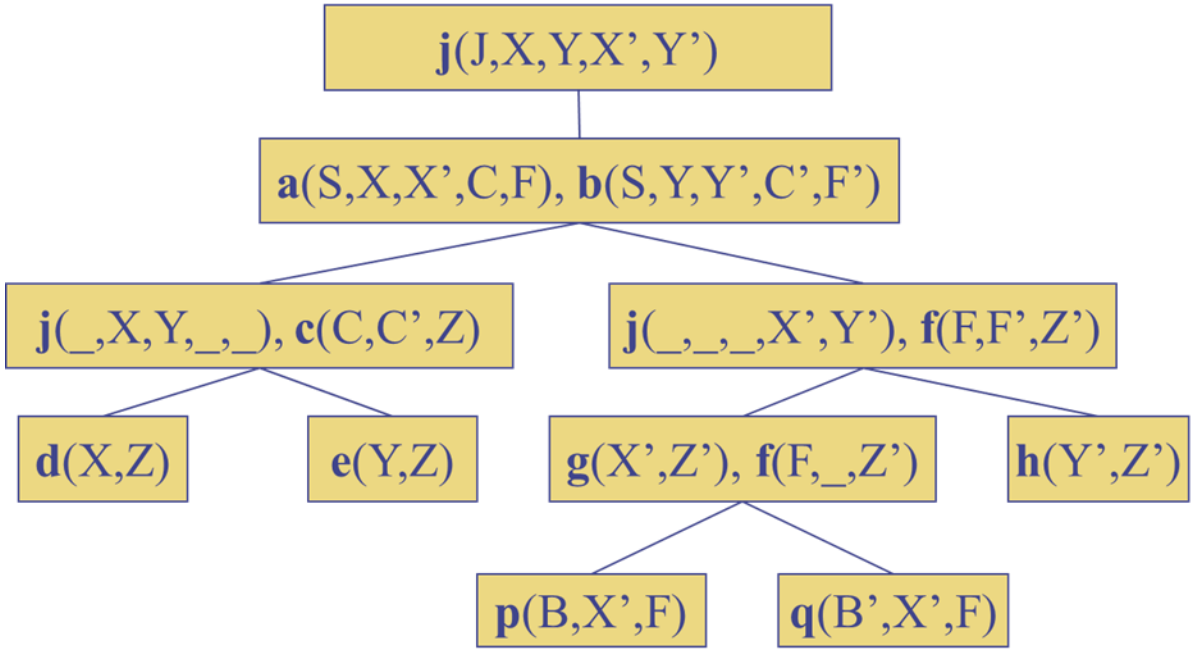
What is hypertreewidth (hw)?

Generalized Hypertree Decomposition = TD + Covering

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$



tw=8



ghw=2

Unfortunately:

GHW is NP-hard to compute, even for small width.

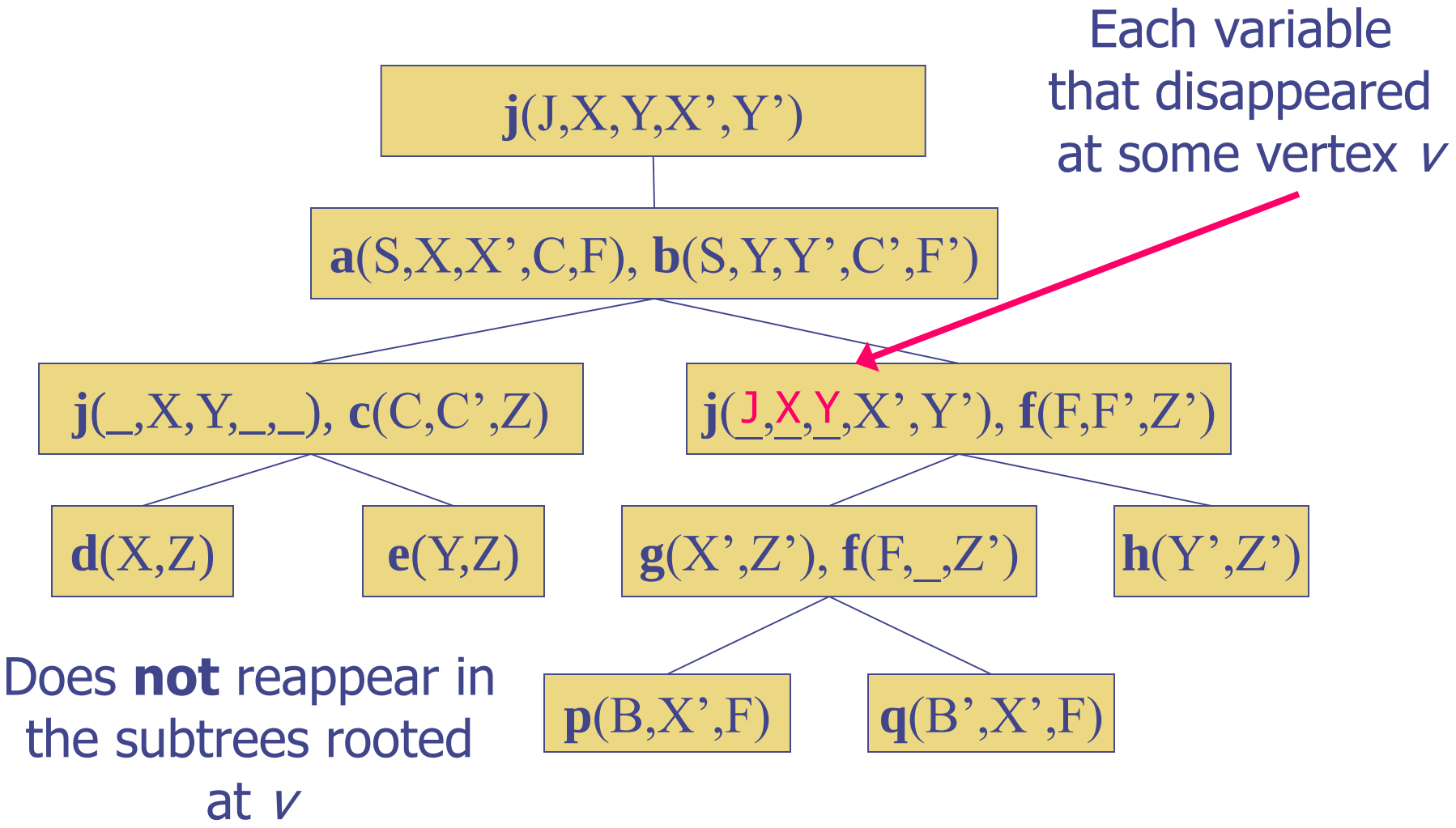
Theorem [G., Miklos, Schwentick 07+09]:

Checking whether ghw(Q)=3 is NP complete

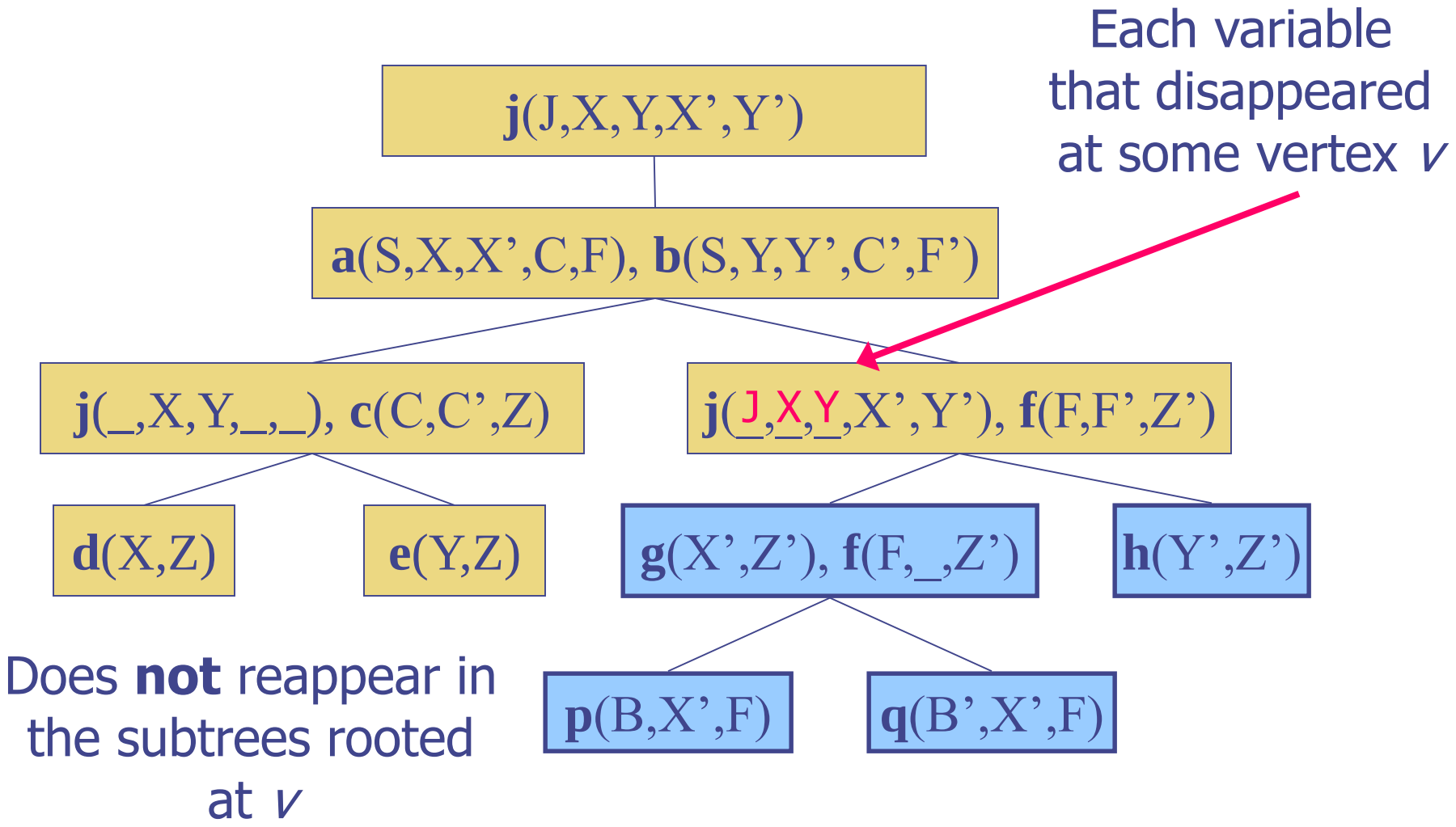
Thus, GHDs do not fulfill criterion 2 (efficient recognizability).

→ Slightly restrict GHDs using a ***special condition***, yielding HDs

Hypertree Decomposition
= GHD +Special Condition



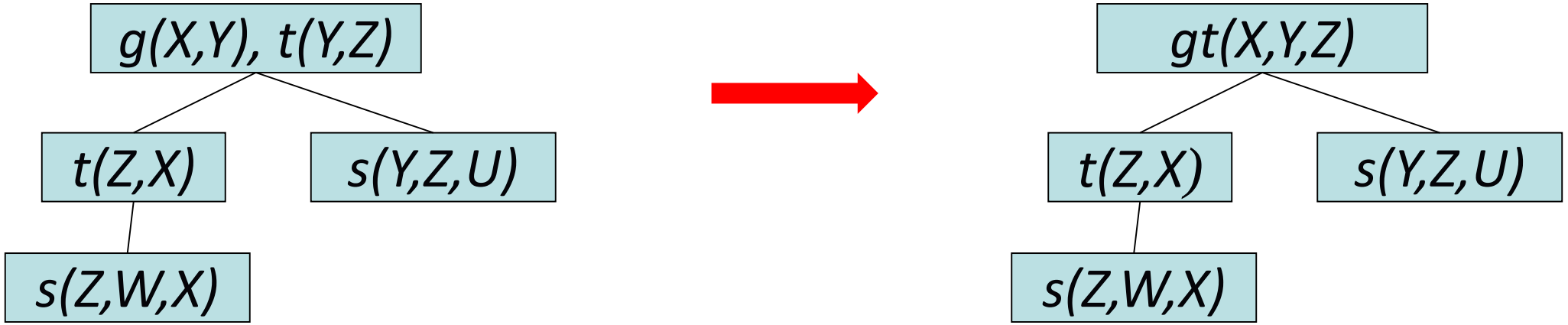
Hypertree Decomposition
= GHD +Special Condition



How to answer queries of bounded GHW or HW by reducing them to acyclic queries?

Transform a query of bounded width into an acyclic query over a modified database

$$q \leftarrow s(Y,Z,U) \wedge g(X,Y) \wedge t(Z,X) \wedge s(Z,W,X) \wedge t(Y,Z)$$



Relations:

Relations:

What are the main good properties of (G)HDs and of queries of bounded (G)HW?

- ✓ GHW and HW generalize acyclicity:
  for an acyclic query *Q*, ghw(Q)=hw(Q)=1 .

- ✓ For fixed *k*, deciding whether
  hw(Q) $\leq$ k is in polynomial time: $O(v \times |Q|^{2k})$.

- ✓ Computing k-hypertree decompositions -if
  they exist- is feasible within the same bound.

- ✓ Answering Boolean CQs of ghw or hw k is
  feasible in time $O(|Q| \times |r_{max}|^k \times \log |r_{max}|)$;
  answering non-Boolean CQs with poly delay.

✓ GHW and HW generalize acyclicity:
for an acyclic query *Q*, ghw(Q)=hw(Q)=1 .

1. generalization

✓ For fixed *k*, deciding whether
hw(Q) $\leq$ k is in polynomial time: $O(v \times |Q|^{2k})$.

✓ Computing k-hypertree decompositions -if they exist- is feasible within the same bound.

2. efficient recognizability

✓ Answering Boolean CQs of ghw or hw k is feasible in time $O(|Q| \times |r_{max}|^k \times \log |r_{max}|)$; answering non-Boolean CQs with poly delay.

3. efficient query-answering

HW, GHW, and TW

✓ GHW and HW improve over treewidth:

$ghw(Q) \leq hw(Q) \leq \frac{1}{2}\, tw(Q) + 1$

✓ Therefore bounded TW → bounded HW and GHW.

But: bounded HW (or GHW) $\not\rightarrow$ bounded TW

How is HW related to GHW?

Approximation Theorem [Adler,G.,Grohe 05] :

$$ghw(Q) \leq hw(Q) \leq 3ghw(Q)+1$$

Thus, a class C of queries has bounded hw iff it has bounded ghw. ✓

In practice, hw and ghw do not differ much.

Do HDs have any disadvantage?

Deciding whether $hw(Q) \leq k$ is in and finding a decomposition is feasible in time $O(v \times |Q|^{2k})$.

Good because polynomial-time.

Bad because high exponent. We won't get rid of it, however:

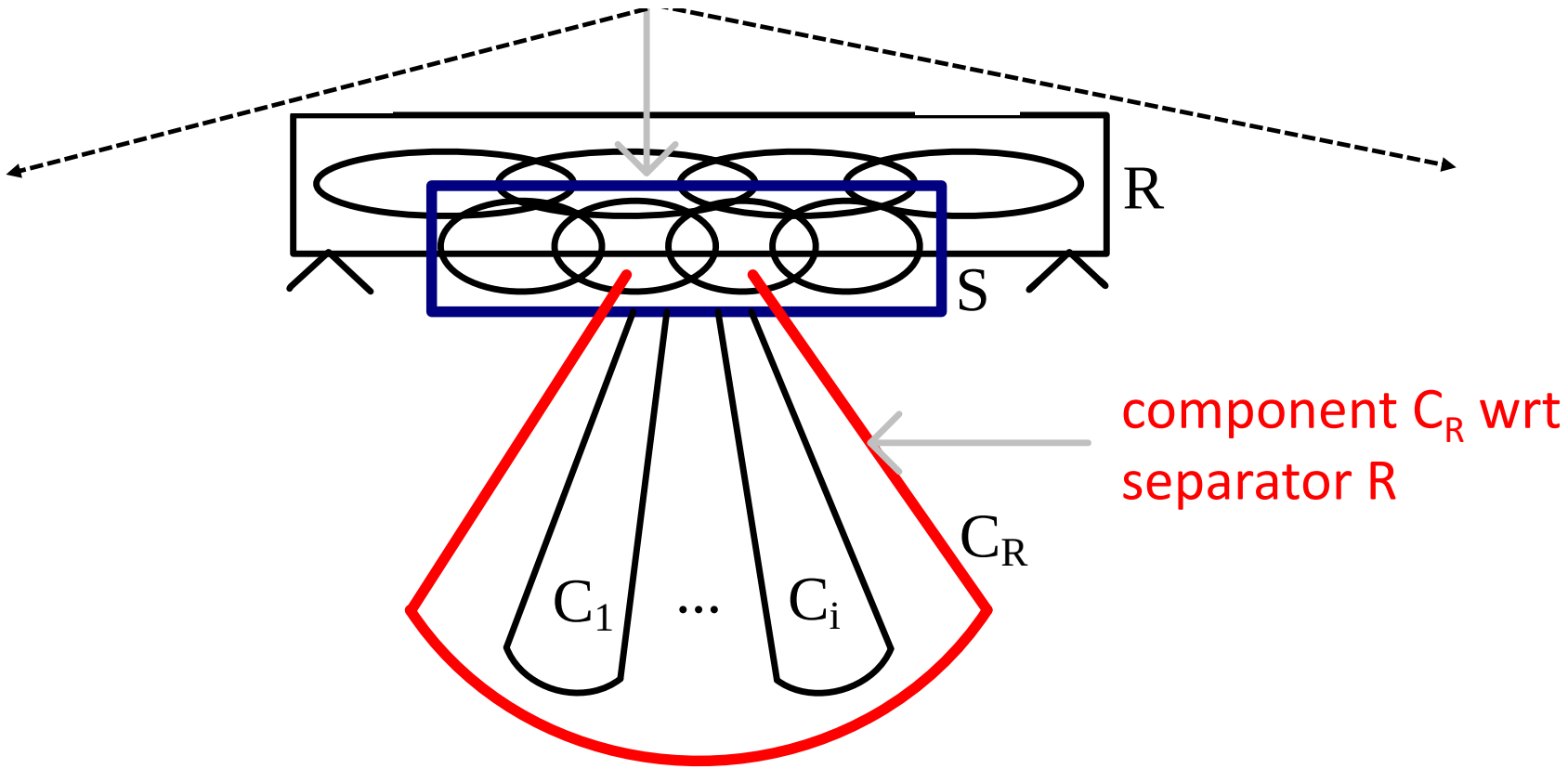
FP-Intractability result [Adler,G.,Grohe 05] :

Checking $hw(Q) \leq k$ is W[2] complete

How are HDs computed?

Polynomial algorithm: Alternating Logspace

Once I have guessed R, how to guess the next bag S ?

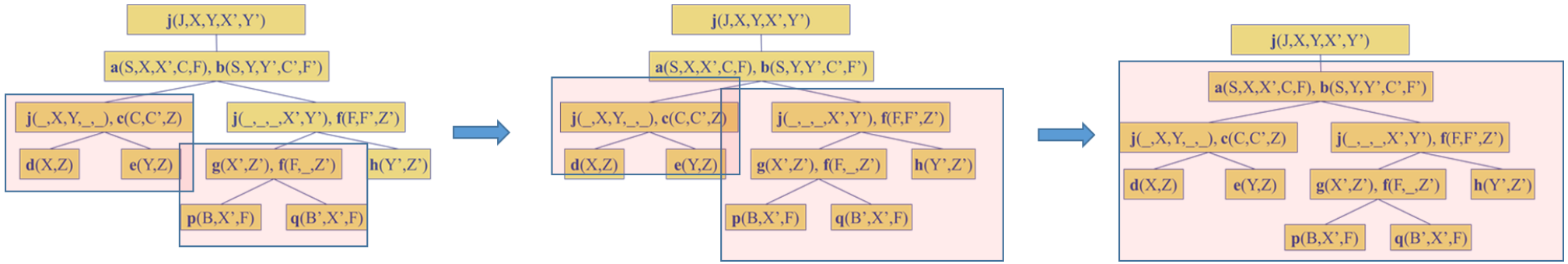


∃-step: choose bag S (polynomially many choices thanks to special condition)

∀-step: Check that for all new components the decomposition can be continued.

ALOGSPACE algorithm → deterministic polytime algorithms

- Opt-k-Decomp: bottom-up construction of HD [G.,Leone,Scarcello 99;01]



- Det-k-Decomp: top-down with backtracking & cashing [G.,Samer 08]
  Better performance due to various optimizations.

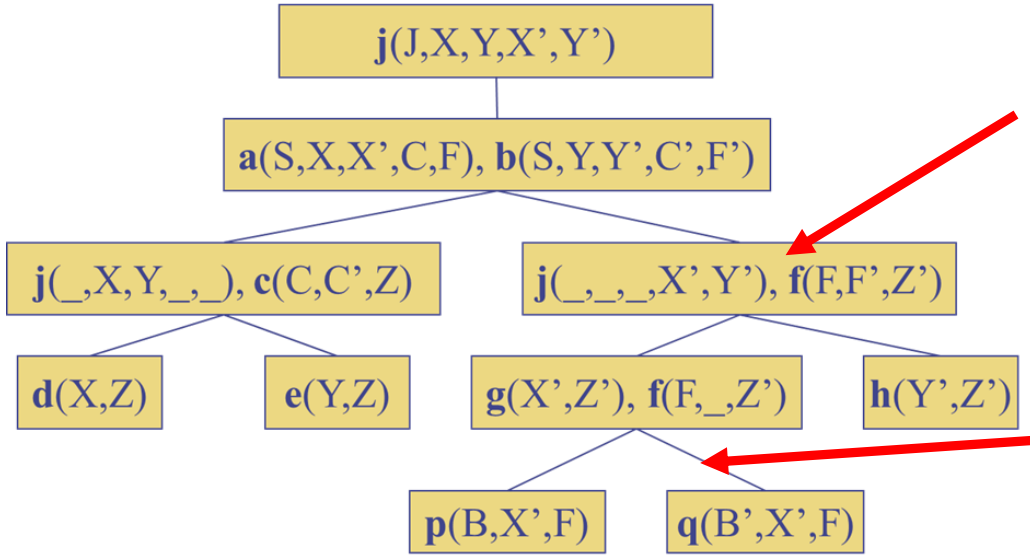
- Heuristic methods for very large hypergraphs [Dermaku et al 05] .

Hypertree Decomposition Homepage: dimes.unical.it/scarcello/Hypertrees

Can HDs be optimized to match physical DB parameters?

(G)HDs are query plans. Based on available selectivity and cardinality indexes, we can associate a cost to each such decomposition:

D:



Local cost c(n) at each decomposition-node n for joins & projections

Local cost c'(n) at each decomposition-edge e for semijoin

$$\text{cost}(D)=\Sigma_{n\in N}\, c(n) \;+\; \Sigma_{e\in E}\, c'(e)$$

Work by Scarcello, Greco, Leone [PODS 04; JCSS 07] :

Theorem: Finding a minimum cost HD (or GHD) is NP-hard.

Note: This is just as bad as classical query optimization.

More surprising: Problem is tractable for a slight restriction of HDs:

Theorem: Finding a minimum cost *nornal-form* HD is tractable.

A k-width HD is in normal form iff it is generated by Opt-k-Dekomp

Algorithm Cost-k-Dekomp

Work by Scarcello, Greco, Leone [PODS 04; JCSS 07] :

Theorem: Finding a minimum cost HD (or GHD) is NP-hard.

Note: This is just as bad as classical query optimization.

More surprising: Problem is tractable for a slight restriction of HDs:

Theorem: Finding a minimum cost *nornal-form* HD is tractable.

A k-width HD is in normal form iff it is generated by Opt-k-Dekomp

Algorithm Cost-k-Dekomp

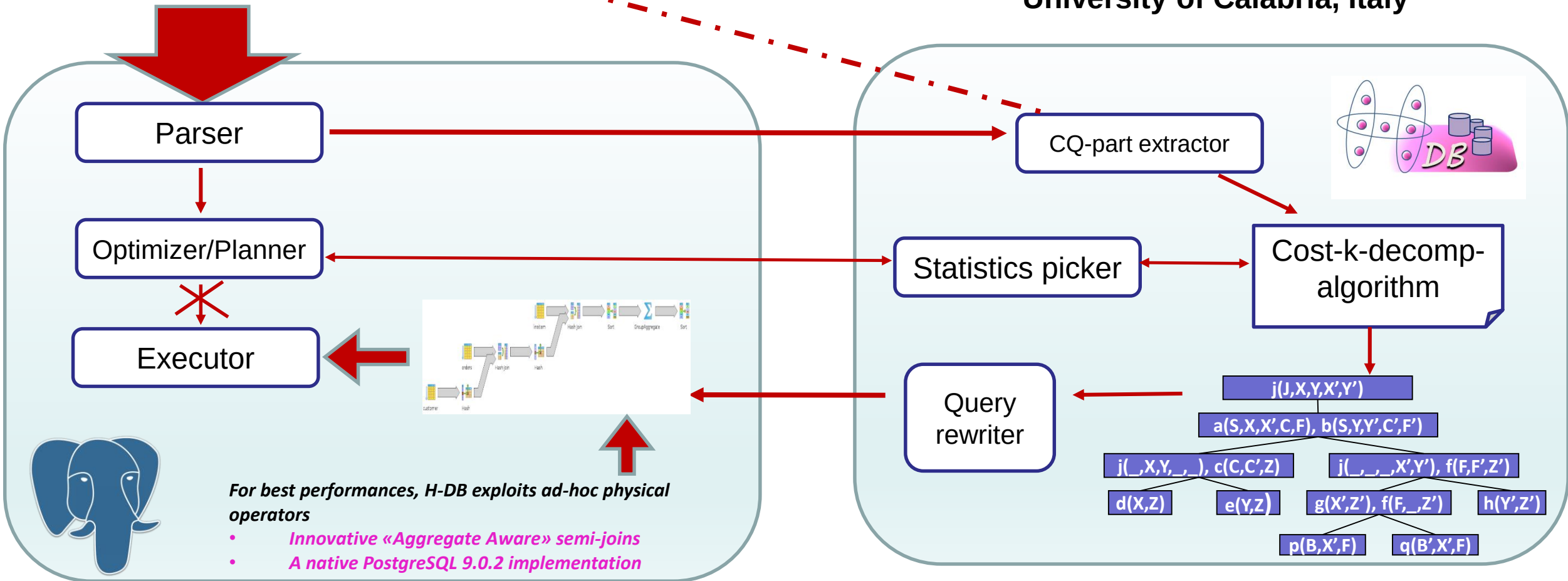
More research needed to understand trade-off

How can HDs be integrated into a DBMS?

Hypertrees for Databases: PostgreSQL boost!

```
--TPCH Query 3
select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriority
from customer, orders, lineitem
where c_mktsegment = 'BUILDING' and c_custkey = o_custkey
      and l_orderkey = o_orderkey and o_orderdate < '1995-03-16'
      and l_shipdate > '1995-03-16'
group by l_orderkey, o_orderdate, o_shippriority;
```

Designed and implemented at University of Calabria, Italy

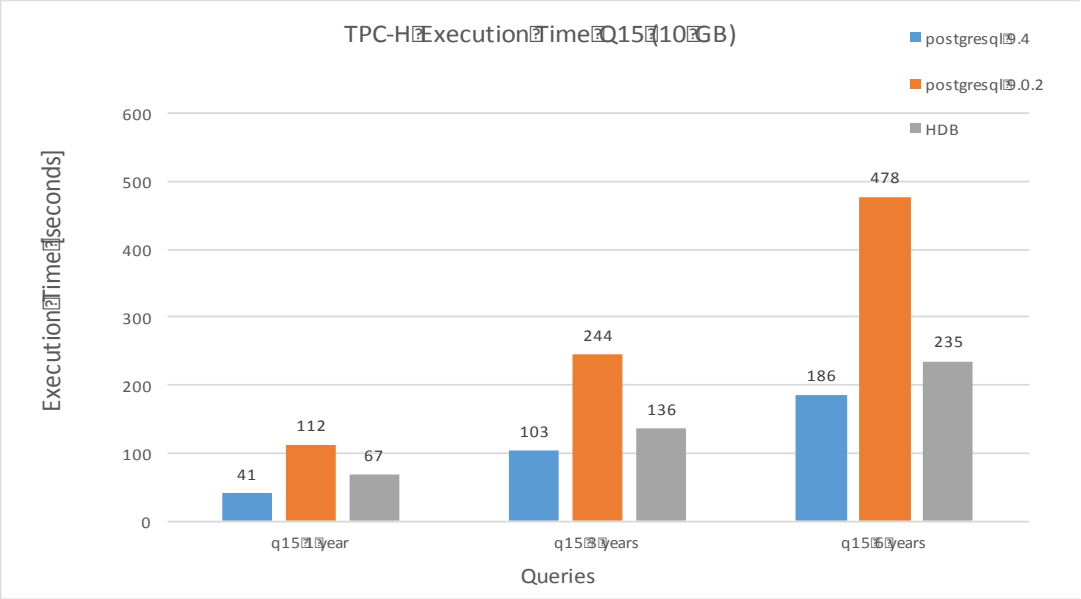


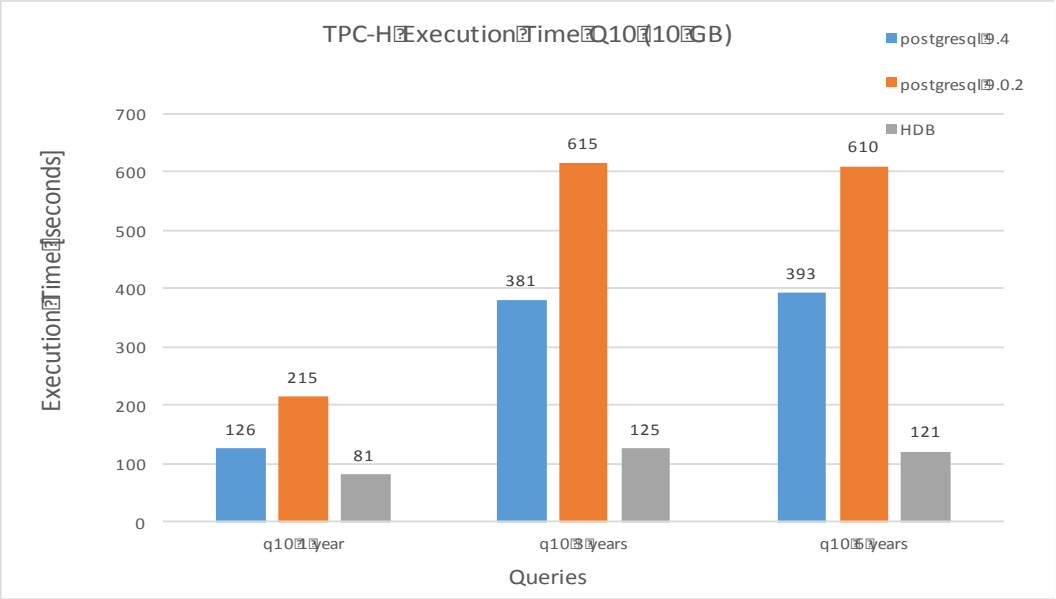
For best performances, H-DB exploits ad-hoc physical operators

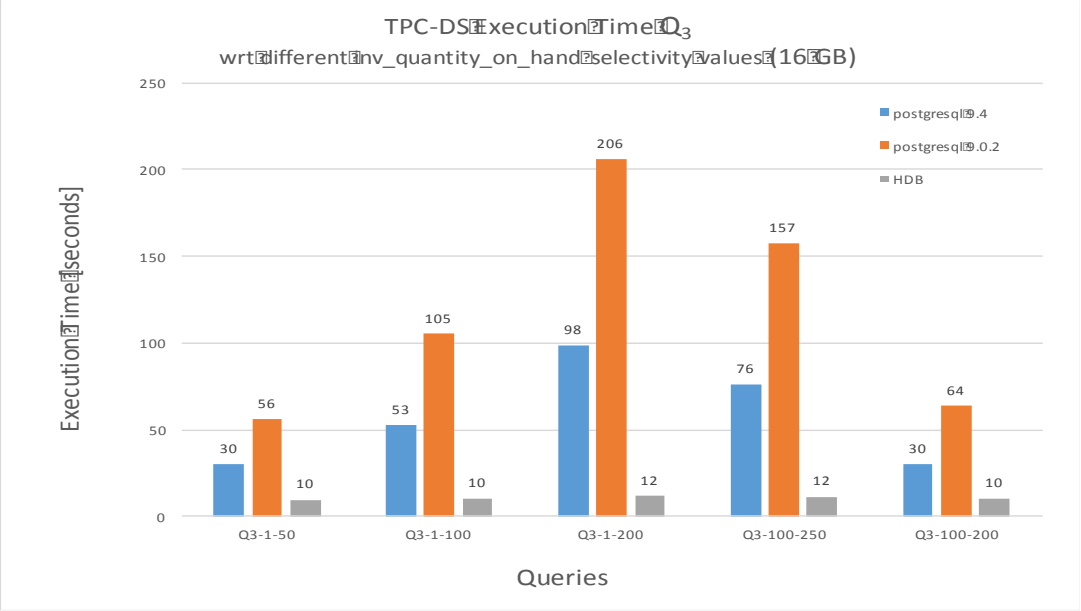
- *Innovative «Aggregate Aware» semi-joins*
- *A native PostgreSQL 9.0.2 implementation*

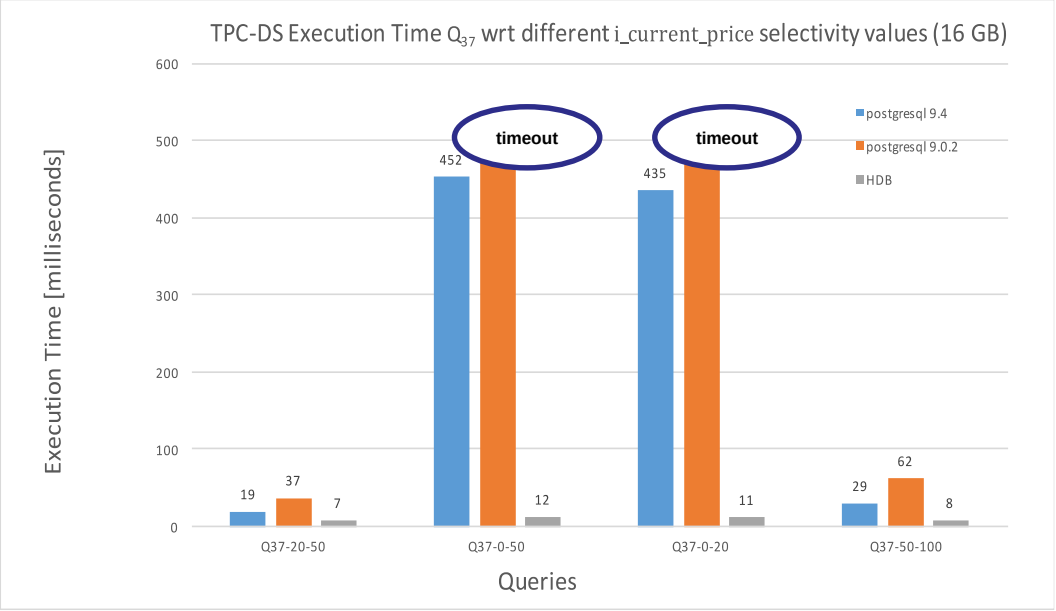
Are HDs useful in practice?

Some experiments (H-DB uses PostgreSQL 9.0.2)









Key issues of implementation

These results tell us that the acyclicity plays a crucial role in this setting which goes well beyond all the gains obtained from the sophisticated quantitative machinery that traditional DBMS performs.

...however, here are some issues that have a negative impact on our performance:

- Due to **major changes** in the structure of the source code of the most recent versions of PostgreSQL (and also for lack of time!!) we were unable to update the implementation of our physical SEMI-JOIN operator.

- But even worse, **PostgreSQL doesn't support "optimizer hints"** as they are implemented in other RDBMS such as MySQL and Oracle.

- As a partial workaround, it is possible to exploit the PostgreSQL query planner trick **OFFSET 0**

- However **OFFSET 0 causes PostgreSQL to fully materialize the subqueries**, by keeping them all in memory (or worse, in temporary disk files) and in some cases (large intermediate results) all the structural optimization benefits are lost .

Recent works implementing GHDs and generalizations:

[Afrati,Joglekar, Ré, Salihoglu,Ullman, CORR 14]:
GYM: A Multiround Join Algorithm In MapReduce.

[Koch , Ahmad, Kennedy, Nikolic, Nötzli, Lupei, Shaikhha VLDBJ 2014]
DBToaster: higher-order delta processing for dynamic, frequently fresh views.

[Tu,Ré SIGMOD 2015]
DunceCap: Query Plans Using Generalized Hypertree Decompositions

[Abo Khamis, Ngo, Rudra PODS 2016 Best Paper]
FAQ: Questions Asked Frequently

[Aberger, Tu, Olukotun, Ré, SIGMOD 2016]
EmptyHeaded: A Relational Engine for Graph Processing.

[Joglekar, Puttagunta,Ré, PODS'16]
AJAR: Aggregations and Joins over Annotated Relations

Can HDs help in parallel and distributed environments?

Theorem [GLS99]: Answering Boolean Queries of bounded HW is LOGCFL-complete

$$AC_0 \subseteq NL \subseteq LOGCFL = SAC_1 \subseteq AC_1 \subseteq NC_2 \subseteq \cdots \subseteq NC = AC \subseteq P \subseteq NP$$

LOGCFL is a class of highly parallelizable problems (on a PRAM)

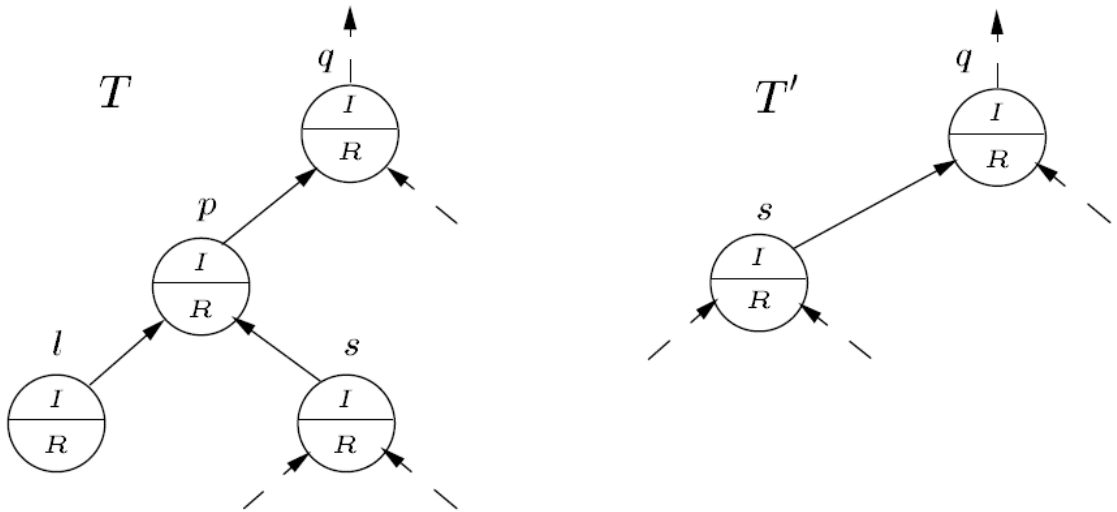
Can be turned into a practical parallel algorithms (for acyclic queries) at the macro-level of relational operations.

[G.,Leone, Scarcello FOCS 98, JACM90] + [G. Leone, Scarcello, TR 98]

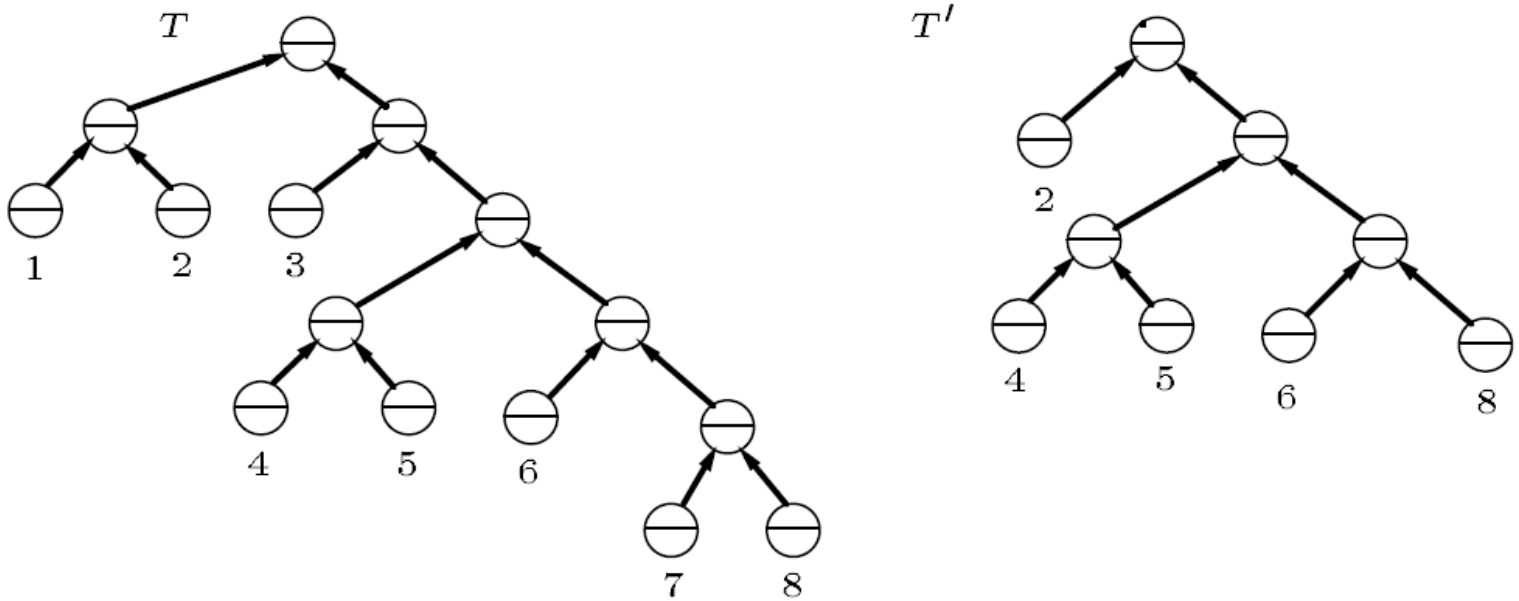
These algorithms clearly apply to queries of bounded HW after the instance has been transformed (in parallel) to an acyclic one.

The gist:

DB-Shunt

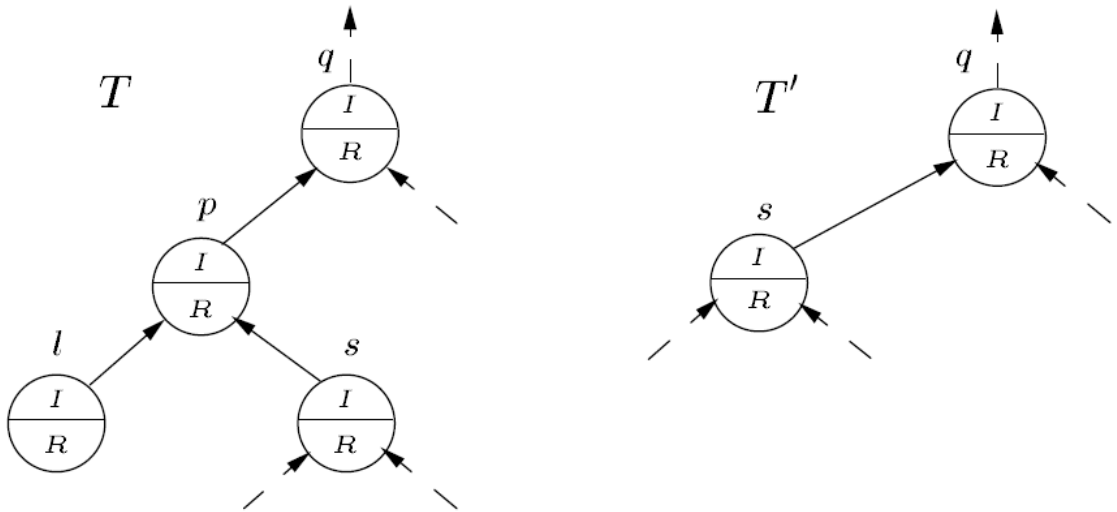


Apply it in parallel to join tree.

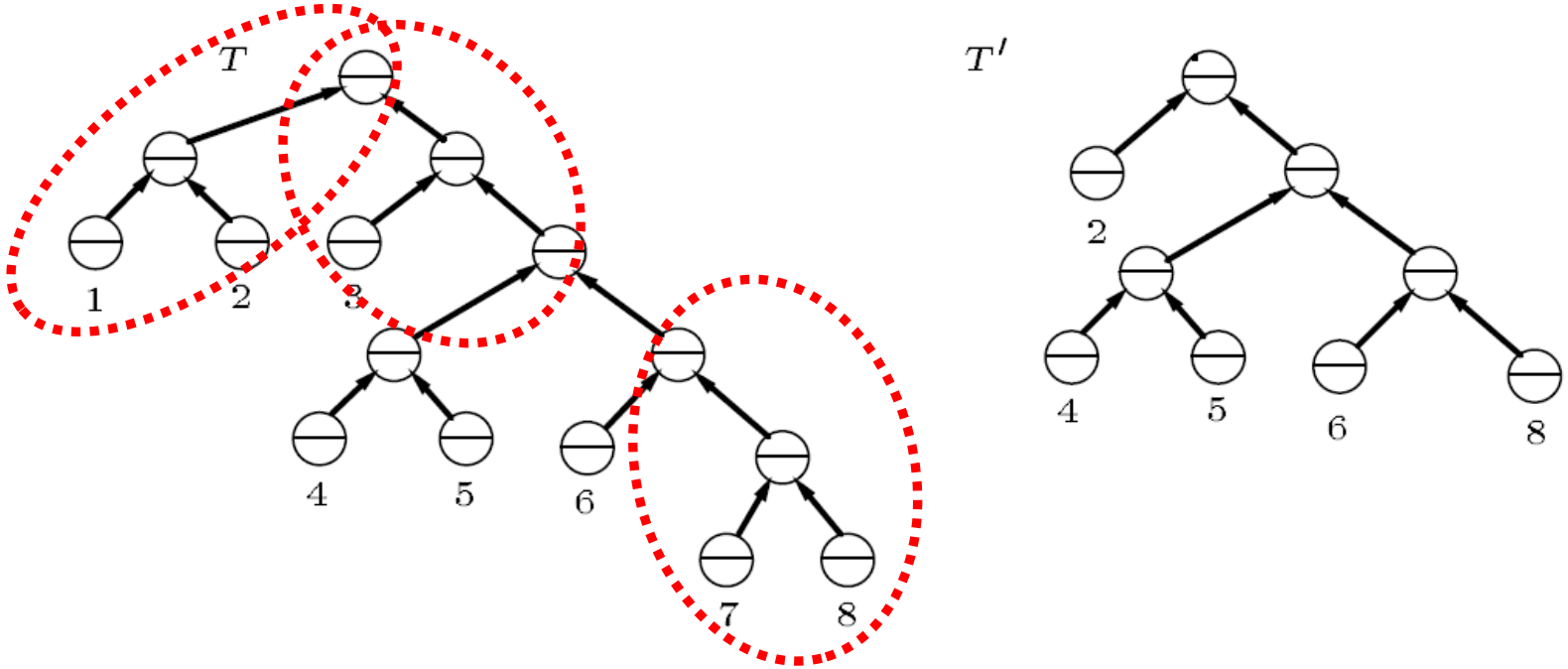


The gist:

DB-Shunt

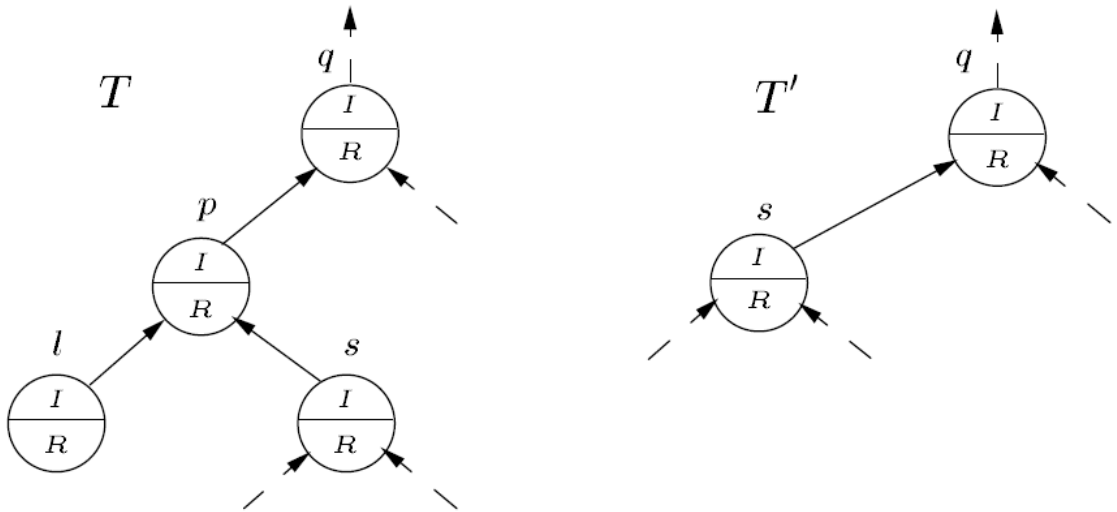


Apply it in parallel to join tree.

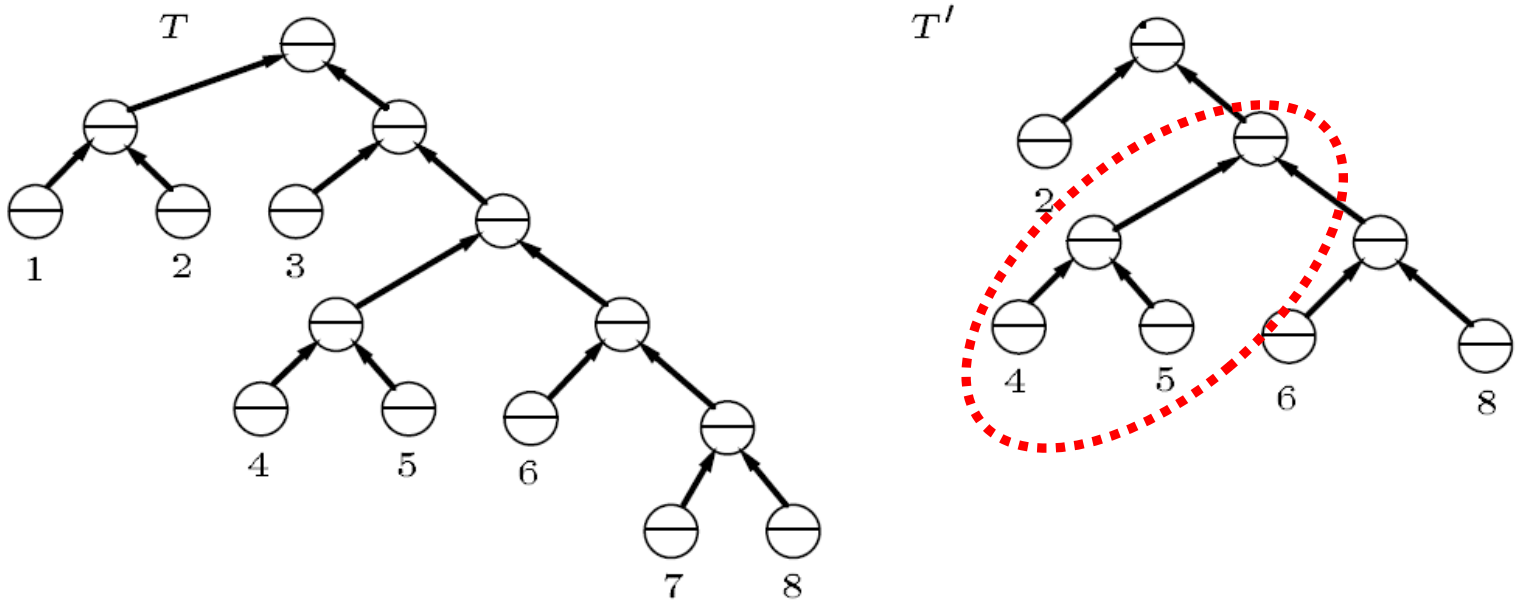


The gist:

DB-Shunt



Apply it in parallel to join tree.



By a similar, but more involved shunt operation, we proved:

Theorem [G.,Leone,Scarcello, Tech Report 1998]:

An acyclic DB instance with n relations can be fully reduced

(a) by a sequence of parallel steps of length O(log n);
(b) by using O(n) processors;
(c) involving O(n) joins in total.

By a similar, but more involved shunt operation, we proved:

Theorem [G.,Leone,Scarcello, Tech Report 1998]:

An acyclic DB instance with n relations can be fully reduced

(a) by a sequence of parallel steps of length O(log n);
(b) by using O(n) processors;
(c) involving O(n) joins in total.

The output to an acyclic query with n atoms can be computed according to (a)-(c), as above, where further

(d) O(n) intermediate relations are created of size $O(\max(d^2, v \times d^2))$

where $d=|r_{max}|$, v= size of output

Based on this approach:

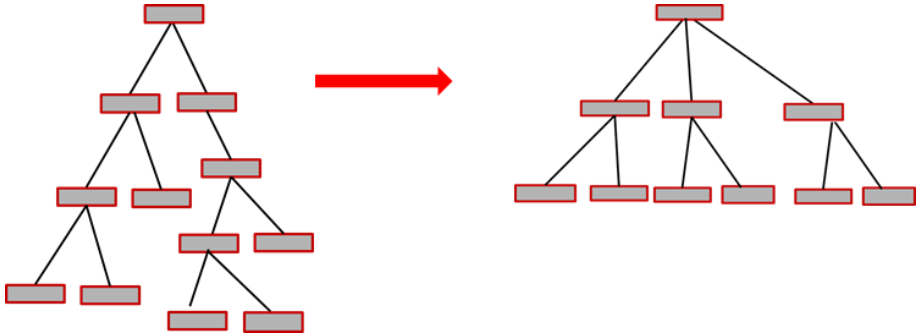
Parallel **ACQ Algorithm** [G,Leone, Scarcello 98].

Multi-round algorithm with O(log |Q|) many rounds

Fits in principle Valiant's BSP model and MapReduce

Alternative approach:

Theorem [Akatov 2010]:
Any HD of width k a query Q can be transformed into a HD of Q of width 3k and depth O(log |Q|).



Theorem [Afrati,Joglekar, Ré, Salihoglu, Ullman, CORR 14]:
The same holds for GHDs

→ **GYM Algorithm**

Can we go beyond conjunctive queries – what about aggregates?

Aggregate Functions for queries of bounded GHW

MAX, TOP-k: tractable for monotone tuple-value functions (+,×)
or, more generally on *smooth evaluation functions* [→GGS 90]

ORDER BY: similar to MAX

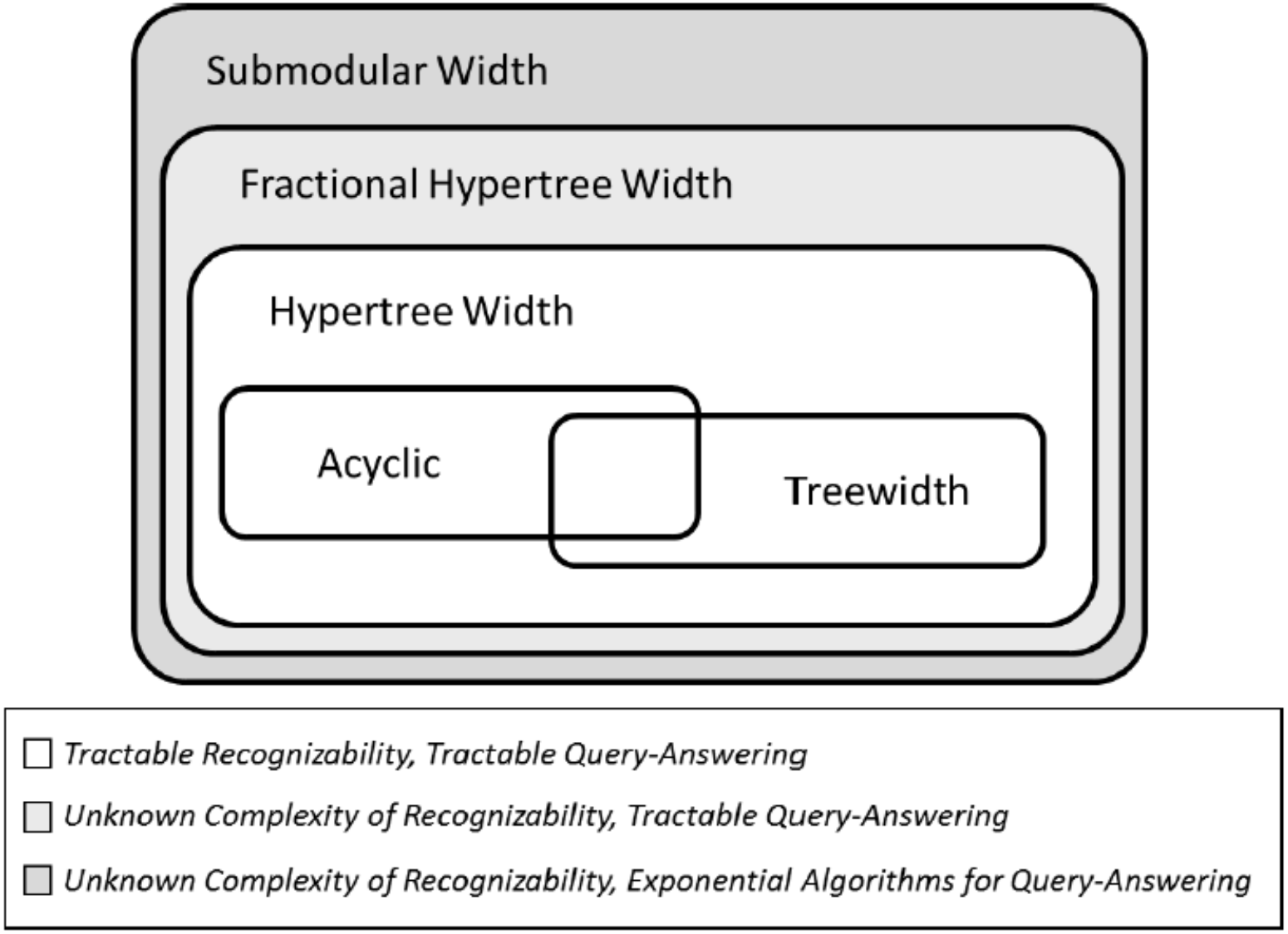
COUNT: #P-hard when not all variables free [Pichler & Skritek 13].
This hardness result holds even for acyclic queries.

Tractable if all variables are free (i.e. are output-vars)
or if other sufficient conditions apply. [GGS 09] [GS 11] [P&S 13]

SUM, AVG: like COUNT

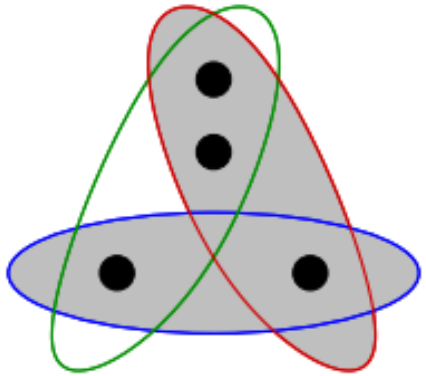
Much more possible: [Abo Khamis, Ngo, Rudra PODS 2016 Best Paper]

Can we go beyond hypertrees?

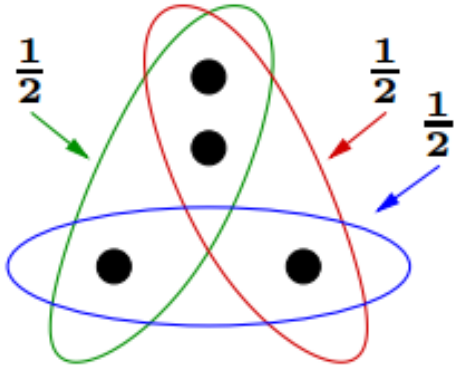

Submodular Width
Fractional Hypertree Width
Hypertree Width
Acyclic
Treewidth
Tractable Recognizability, Tractable Query-Answering
Unknown Complexity of Recognizability, Tractable Query-Answering
Unknown Complexity of Recognizability, Exponential Algorithms for Query-Answering

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.

$\varrho^*(H)$: smallest total weight of a fractional edge cover.



$\varrho(H) = 2$



$\varrho^*(H) = 1.5$

© [Marx 2005]

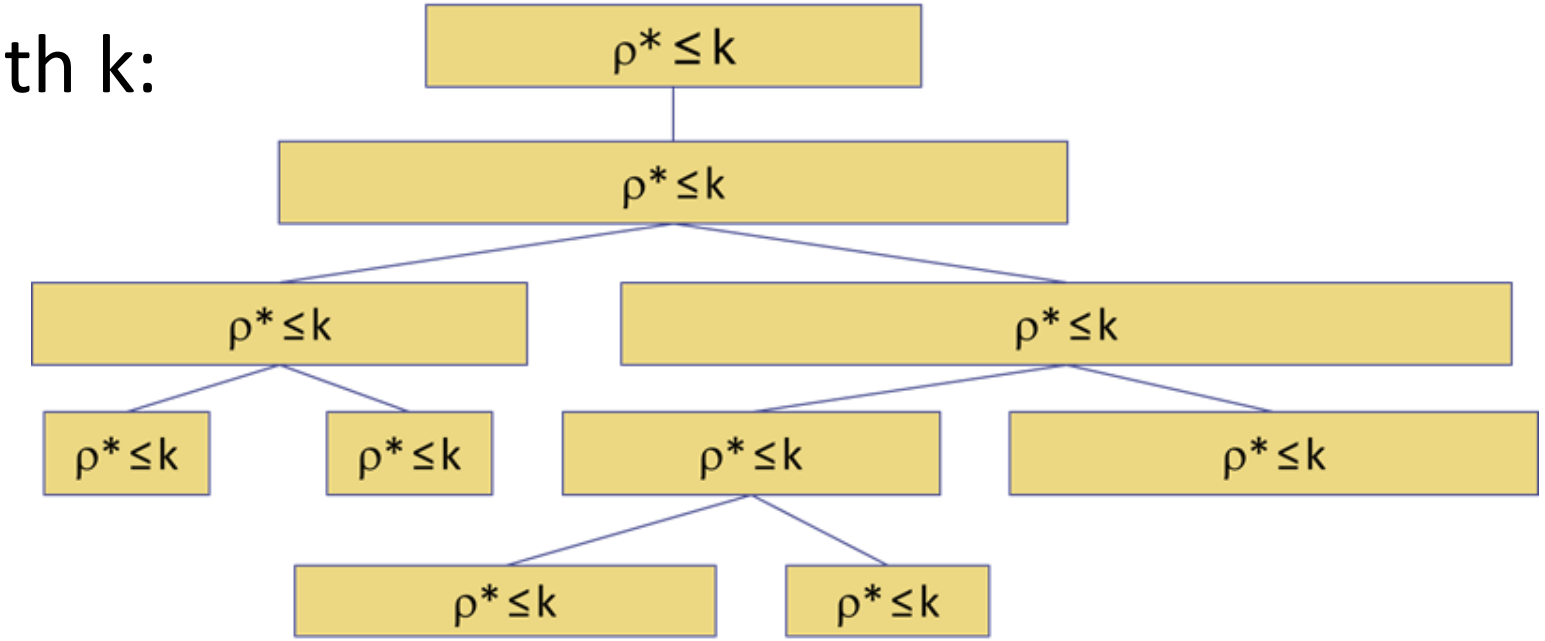
Theorem [Grohe & Marx 06] : The answer to a query of fractional cover weight $\rho^*(Q)$ can be computed in time $|Q| \times rmax^{\rho^*(Q)+O(1)}$

Observe: $\{Q \mid \rho^*(Q) \leq k\}$ and $\{Q \mid hw(Q) \leq k\}$ are incomparable.

To combine the two notions profitably, Grohe and Marx defined

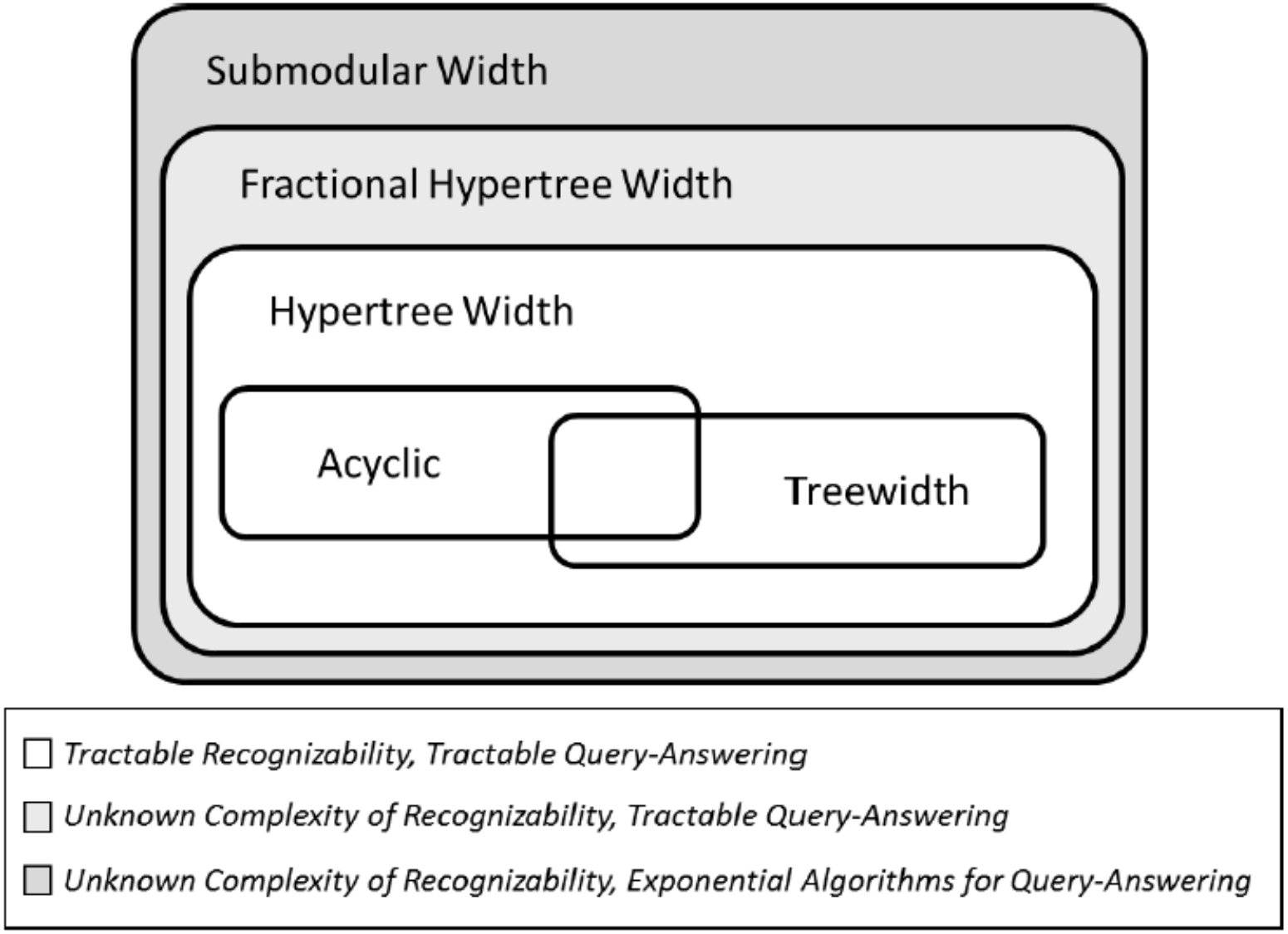
Fractional Hypertree Decompositions (FHDs) and correspondingly FHW

FHD of width k:



Facts about FHDs and FHW

- FHDs properly generalize FHDs, in particular $fhw(Q) \leq ghw(Q) \leq hw(Q)$.
- There are classes of queries of bounded FHW and unbounded GHW.
- Queries of bounded FHW can be answered in polynomial time by transforming the FHD in polytime into an acyclic query (as for HDs).
- It is unknown whether checking $fhw(Q)=k$ for fixed k is tractable.
- Approximation: If $fhw(Q)=w$, we can find an FHD of width $O(w^3)$


Submodular Width
Fractional Hypertree Width
Hypertree Width
Acyclic
Treewidth
Tractable Recognizability, Tractable Query-Answering
Unknown Complexity of Recognizability, Tractable Query-Answering
Unknown Complexity of Recognizability, Exponential Algorithms for Query-Answering

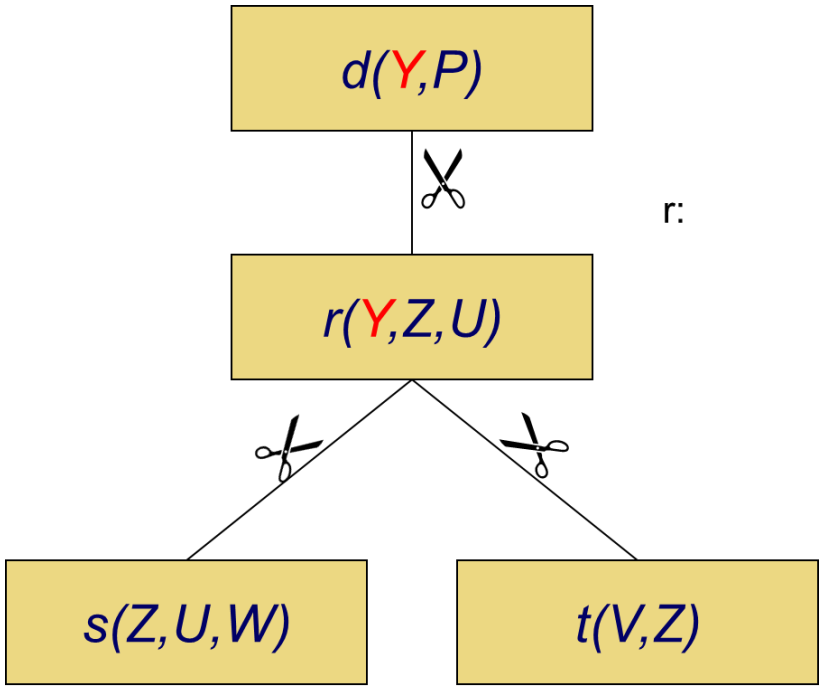
To our best knowledge, hypertreewidth, introduced at PODS 1999, is (up to trivial variants) still the only decomposition method known to fulfill the 3 criteria:

1. *generalization of acyclicity,*
2. *efficient recognizability,*
3. *efficient query-answering.*

Is it actually necessary to compute a decomposition?

Let‘s illustrate this with acyclic queries. For HW k this is similar.

Imagine we do not have a join tree, but we know the query is acyclic (promise problem)



As shown by Chen and Dalmau [CP 2005],we can still fully reduce the query relations by using a polynomial number of semijoins.

We give a simple illustration of this.

Apply the following algorithm:

for 1 to 2(|Q|-1) do:
perform all |Q|×(|Q|-1) possible semijoins.
end.

This is clearly a full reducer. Also works for „semantically acyclic queries“. But uses $O(|Q|^3)$ SJs.

For semanticx acyclicity: → talk [Barcelo, G., Pieris, PODS 16]

What are hot issues for future research?

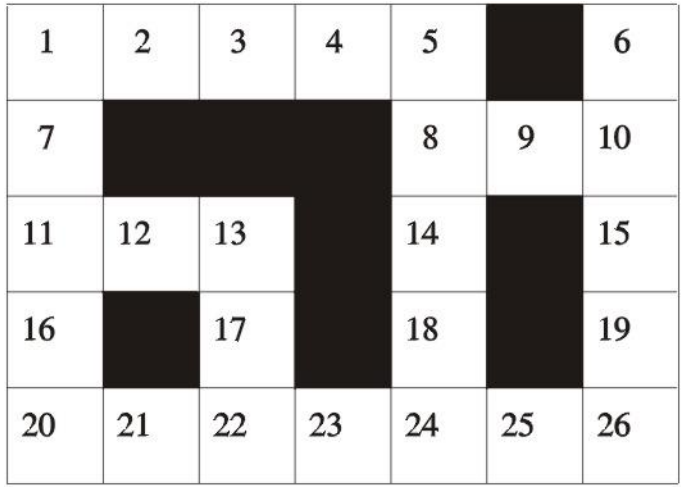
- Are there more general decomposition methods fulfilling (1)-(3)?
- Can can one check FHD(Q)=k in polynomial time for fixed k?
- Are there faster methods to compute HDs (2k in the exponent)
- Find better heuristic methods for computing HDs.
- Can minimum cost HDs be well approximated?
- Better integrate classical query optimization methods with HDs.
- Analyse optimization through HDs on large benchmarks.

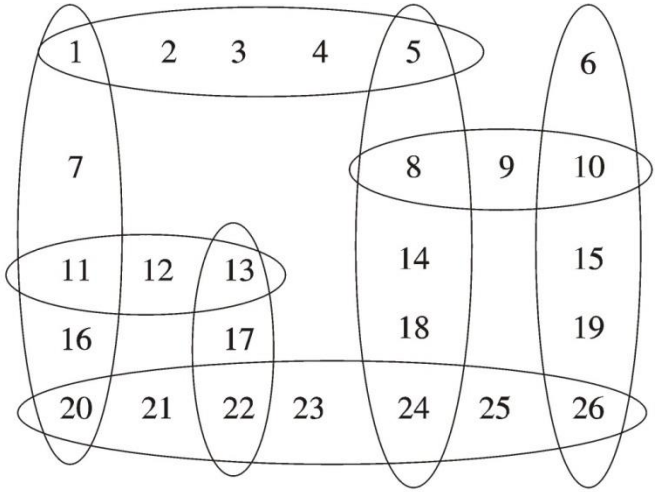
Thank You

APPENDIX

Can HDs be applied outside the DB Domain?

Example of CSP: Crossword Puzzle





1h:

P A R I S
P A N D A
L A U R A
A N I T A

1v:

L I M B O
L I N G O
P E T R A
P A M P A
P E T E R

and so on

Constraint satisfaction problems: Renault example(1/2)

- Renault Megane configuration *[Amilhastre, Fargier, Marquis AIJ, 2002]* Used in CSP competitions and as a benchmark problem

- Variables encode type of engine, country, options like air cooling, etc.

- The considered instances consist of about **150 atoms/constraints** and **110 variables**, with database instances where **attributes** have at most **42** distinct values, and the largest constraint relation contains **48721 tuples**.

Constraint satisfaction problems: Renault example(2/2)

- We discovered that the generalized hypertree width is 3 for most instances (with a maximum of 4).
- The total number of solutions is about $2 \cdot 10^{12}$, however this information is not very meaningful, because of the many auxiliary variables occurring in the problem.
- Rather, by using the algorithms based on generalized hypertree decompositions, it is possible to compute the solutions of these instances (or just their number) over the actual variables of interest.
- None of the other available engines that we know, either in the database or in the CSP community, were able to compute such a result for those large instances.

For more information, we refer to the Hypertree Decomposition web-page:
http://www.dimes.unical.it/scarcello/Hypertrees/

Combinatorial Auctions

Bidders can place bids on
Packages of items.

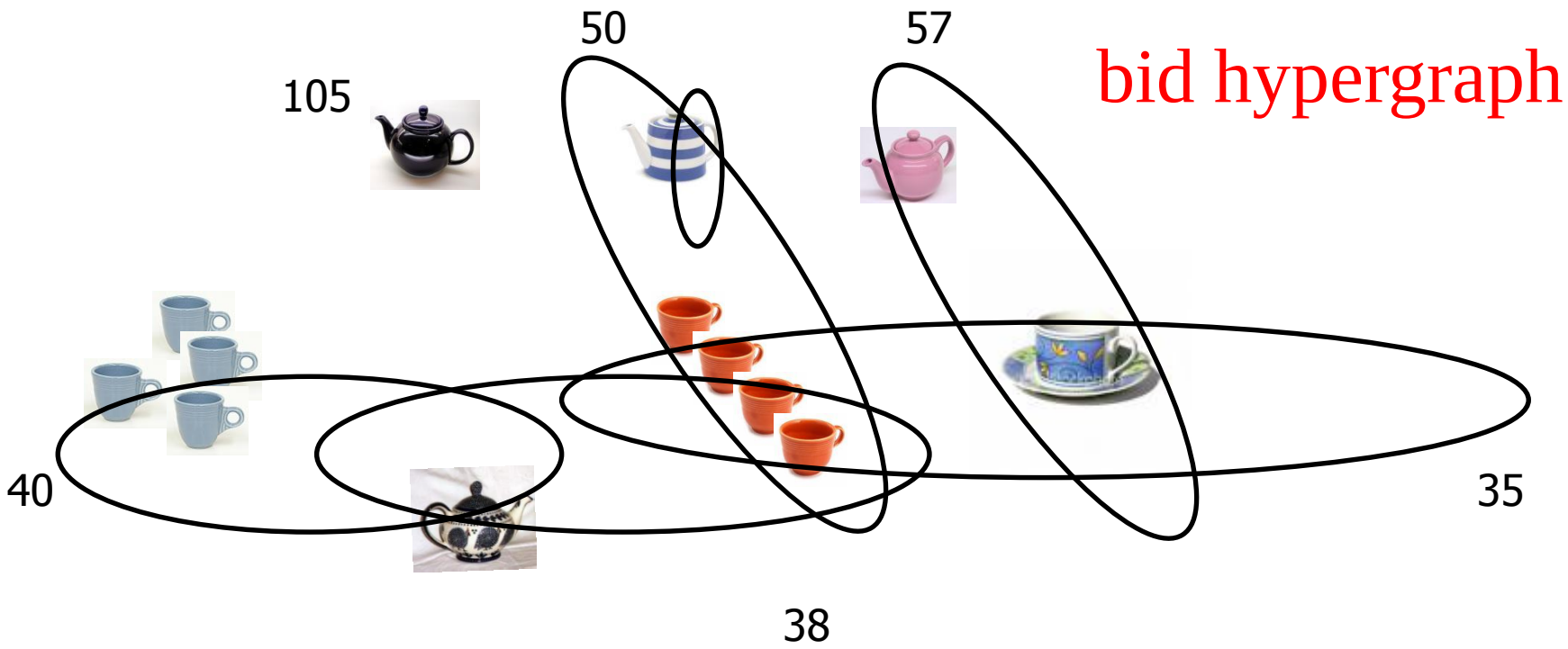
<u>Winner determination:</u> Choose a set of compatible bids of maximum revenue or minimum cost.

For classical auctions, winner determination is obviously tractable. Not so for CAs.

Combinatorial Auctions

Combinatorial Auctions

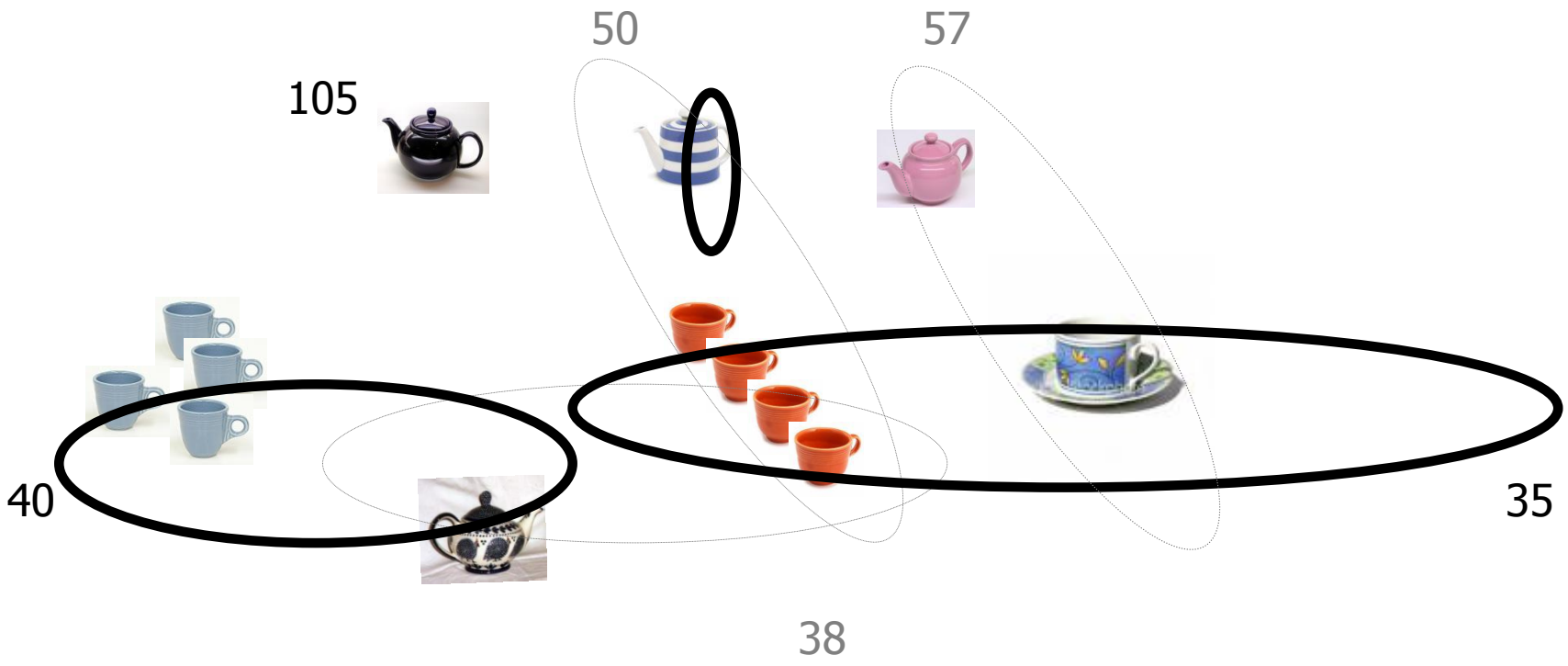
Combinatorial Auctions



Winner determination is intractable (NP-hard)

≡ weighted set packing problem

Combinatorial Auctions

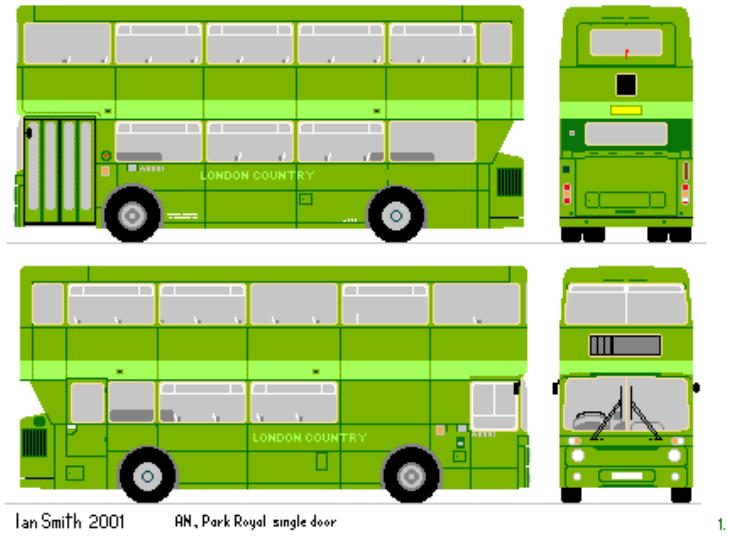


Winner determination is intractable (NP-hard)

≡ weighted set packing problem

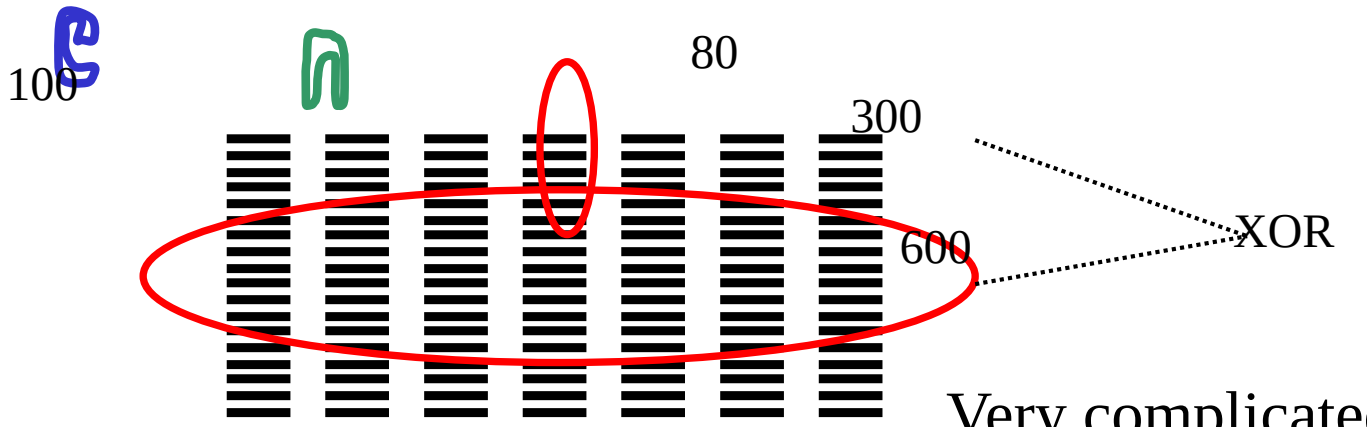
Total £ 180.--

Applications in different domains



London Regional Transports:
Combinatorial auctions of bus routes.
Private bus companies bid on bundles of routes.

Airport slot auction



Very complicated hypergraphs!

Can we decompose and simplify combinatorial auctions?

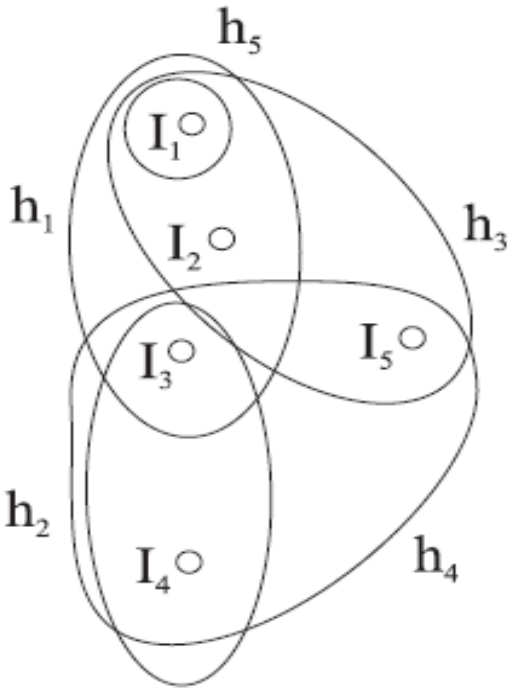
- Has been investigated for many years
- A decomposition method was proposed:
  Structured Item Graph (SIG)

  [Conitzer & Sandholm 2004]

  - If small width SIG exists, then winners can be determined efficiently.
  - Can small-width SIGs be efficiently determined?

Theorem: Determining if an auction admits SIGs of width ≤ 3 is intractable (NP cplt.)

[G. & Greco, ACM Conf on EC 2007]



item hypergraph

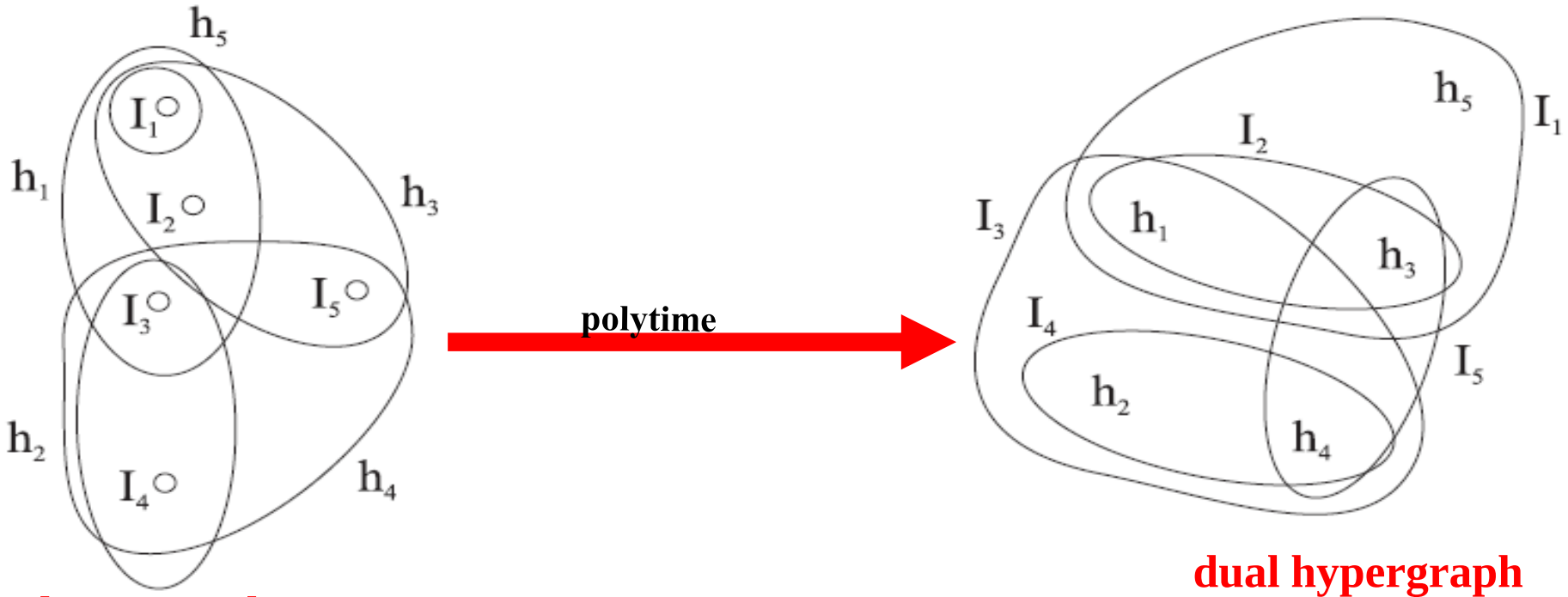
Hypertree decompositons cannot be used directly on the item hypergraph.

The problem remains hard even in case of small-width decompositions

But they can be used on the *dual* hypergraph!

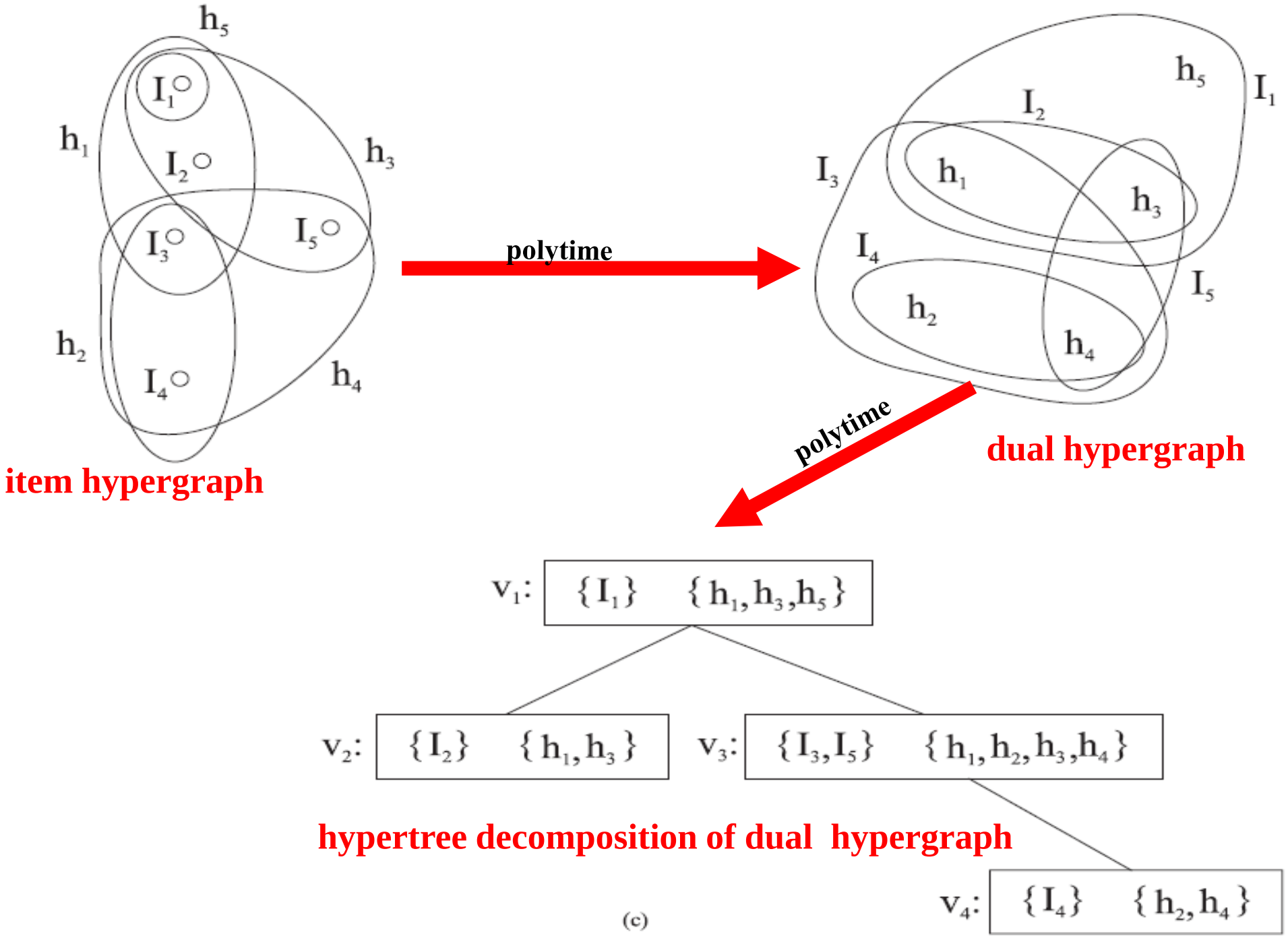
Theorem: The winner determination problem can be solved efficiently if the dual hypergraph has bounded hypertree width. [G.&Greco, 2007]

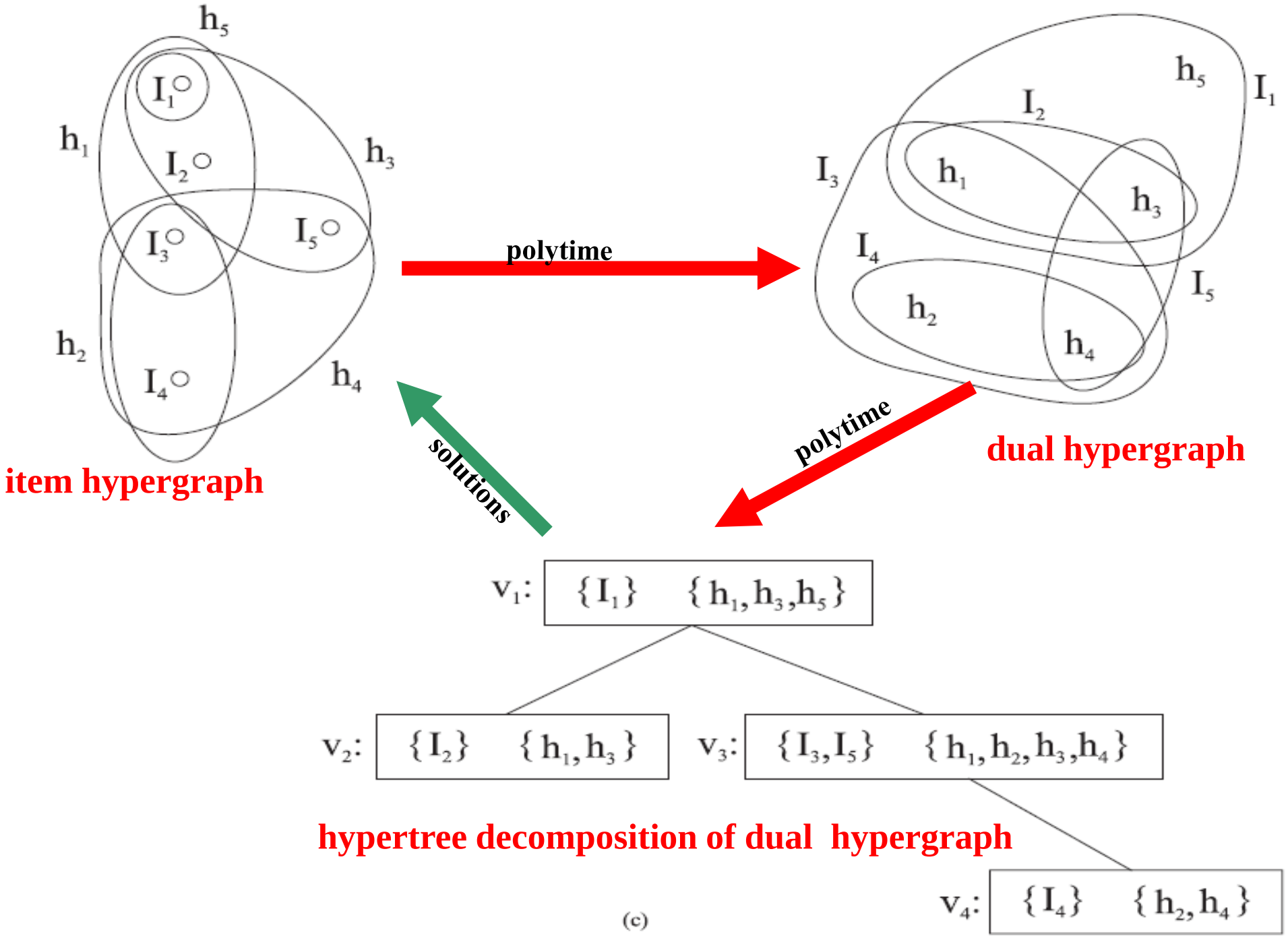
In practice, this is often the case!



item hypergraph

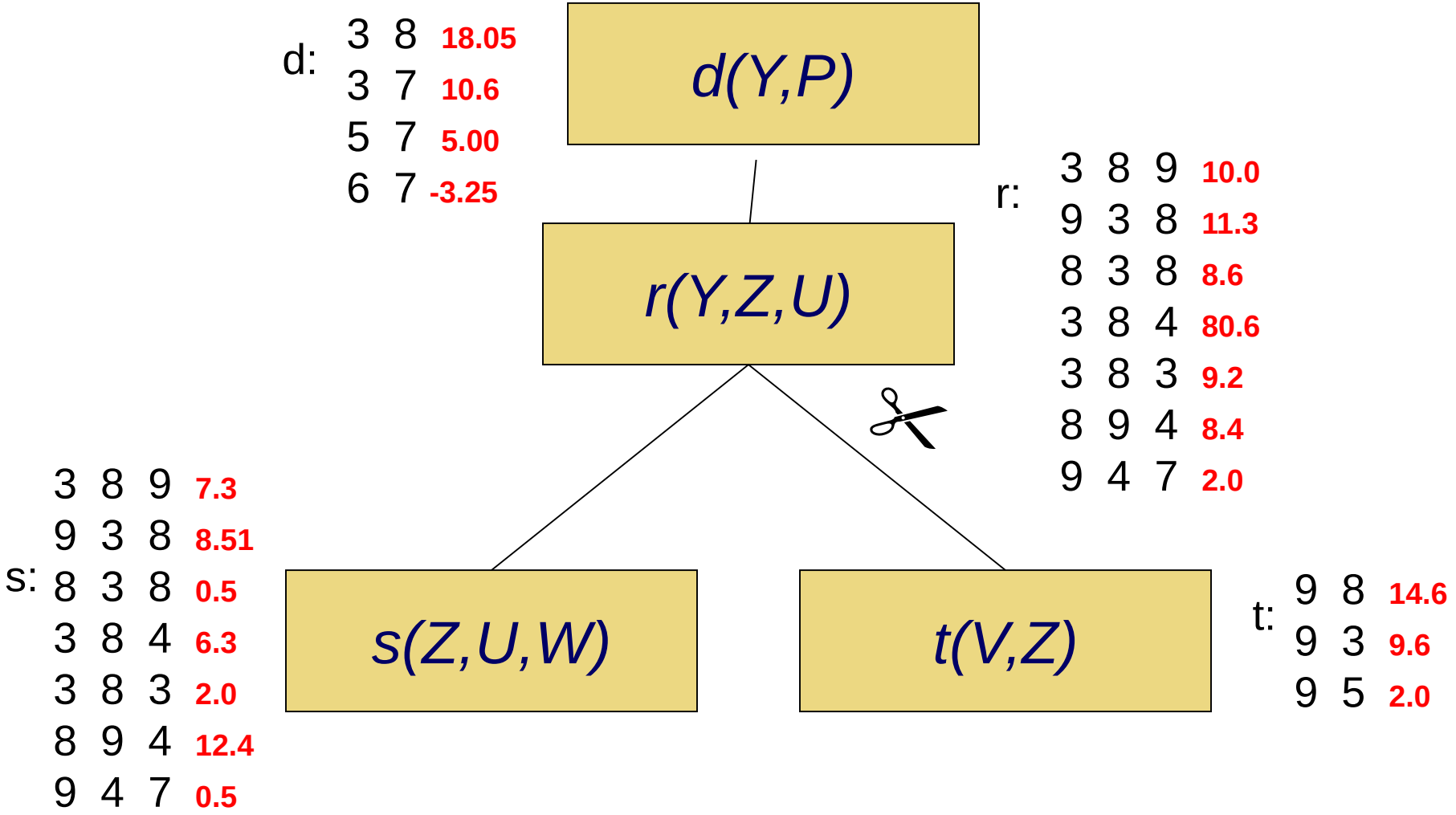
dual hypergraph


h5
I1
h1
I2
h3
I3
I5
h2
I4
h4
item hypergraph
polytime
h5
I1
I2
I3
h1
h3
I4
I5
h2
h4
dual hypergraph
polytime
v1: {I1} {h1,h3,h5}
v2: {I2} {h1,h3}
v3: {I3,I5} {h1,h2,h3,h4}
hypertree decomposition of dual hypergraph
v4: {I4} {h2,h4}
(c)

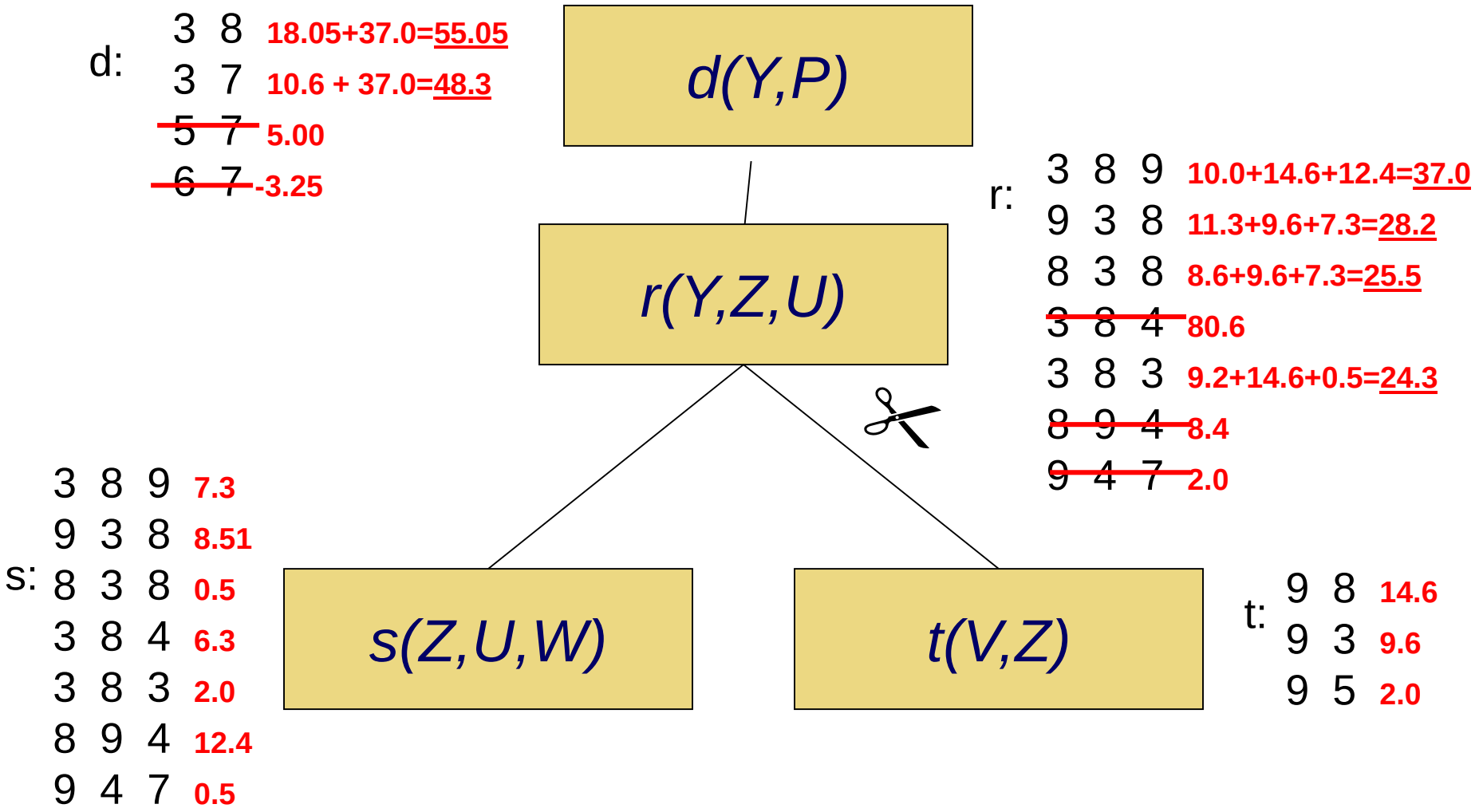

h5
I1
h1
I2
h3
I3
I5
h2
I4
h4
item hypergraph
polytime
I2
h5
I1
I3
h1
h3
I4
I5
h2
h4
dual hypergraph
solutions
polytime
v1: {I1} {h1,h3,h5}
v2: {I2} {h1,h3}
v3: {I3,I5} {h1,h2,h3,h4}
hypertree decomposition of dual hypergraph
v4: {I4} {h2,h4}

(c)

Optimization Version



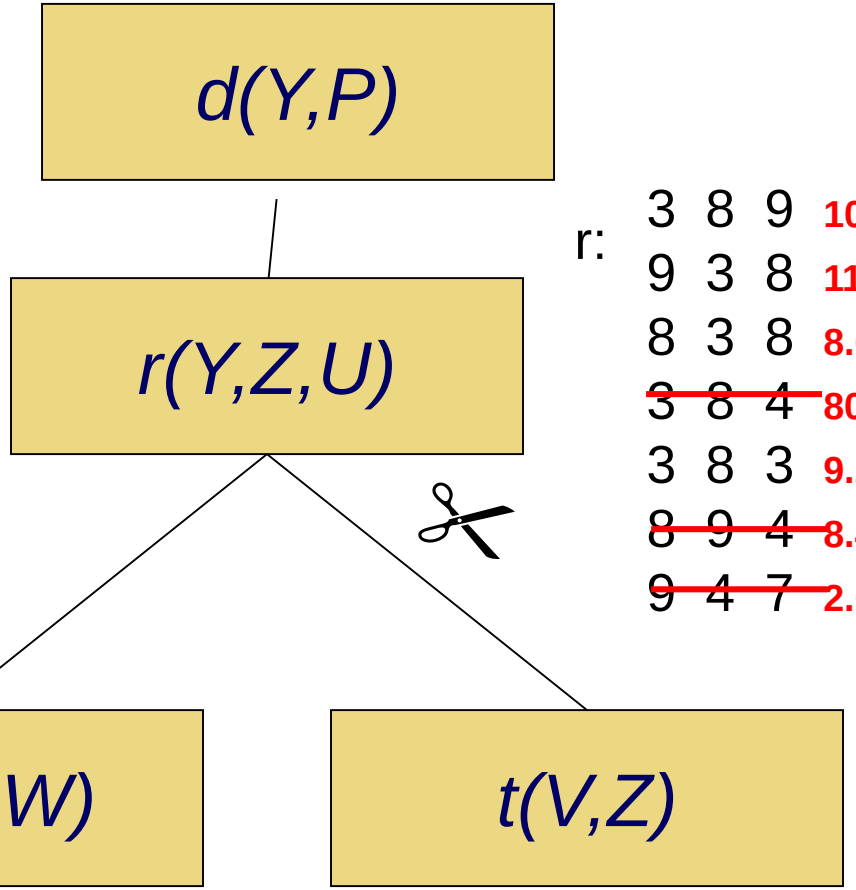
Optimization Version



Optimization Version

d:

3	8
3	7
~~5~~	~~7~~
~~6~~	~~7~~



r:

3	8	9	10.0+14.6+12.4=37.0
9	3	8	11.3+9.6+7.3=28.2
8	3	8	8.6+9.6+7.3=25.5
~~3~~	~~8~~	~~4~~	80.6
3	8	3	9.2+14.6+0.5=24.3
~~8~~	~~9~~	~~4~~	8.4
~~9~~	~~4~~	~~7~~	2.0

s:

3	8	9	7.3
9	3	8	8.51
8	3	8	0.5
3	8	4	6.3
3	8	3	2.0
8	9	4	12.4
9	4	7	0.5

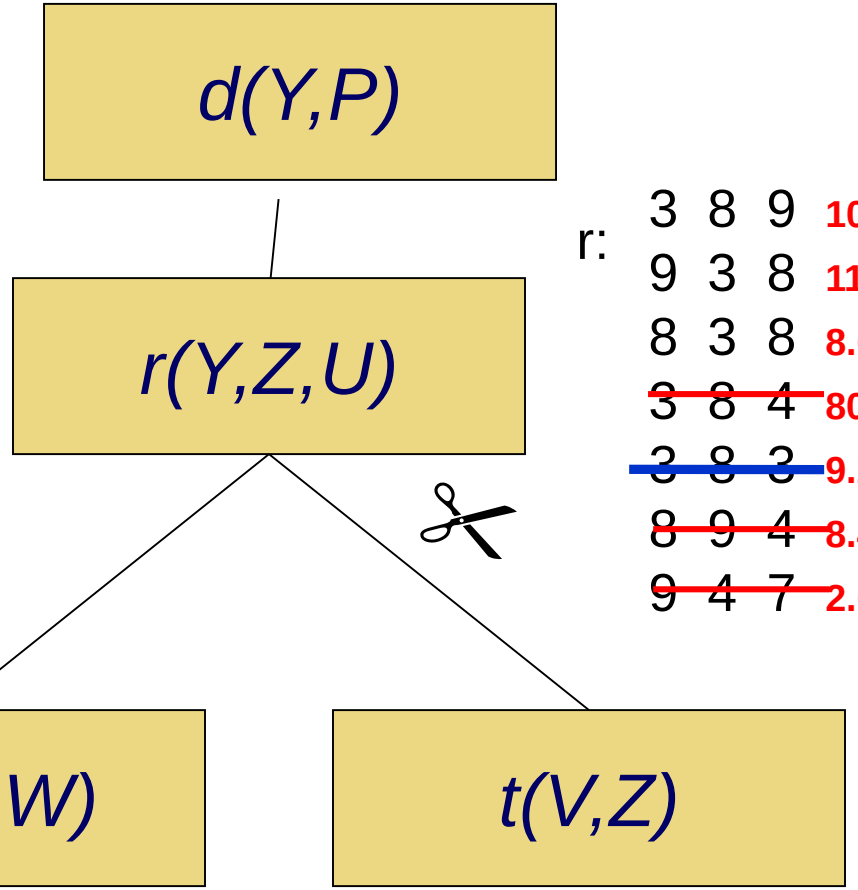
t:

9	8	14.6
9	3	9.6
9	5	2.0

Optimization Version

d:

3	8	18.05+37.0=55.05
3	7	10.6 + 37.0=48.3
~~5~~	~~7~~	5.00
~~6~~	~~7~~	-3.25



r:

3	8	9	10.0+14.6+12.4=37.0
9	3	8	11.3+9.6+7.3=28.2
8	3	8	8.6+9.6+7.3=25.5
~~3~~	~~8~~	~~4~~	80.6
~~3~~	~~8~~	~~3~~	9.2+14.6+0.5=24.3
~~8~~	~~9~~	~~4~~	8.4
~~9~~	~~4~~	~~7~~	2.0

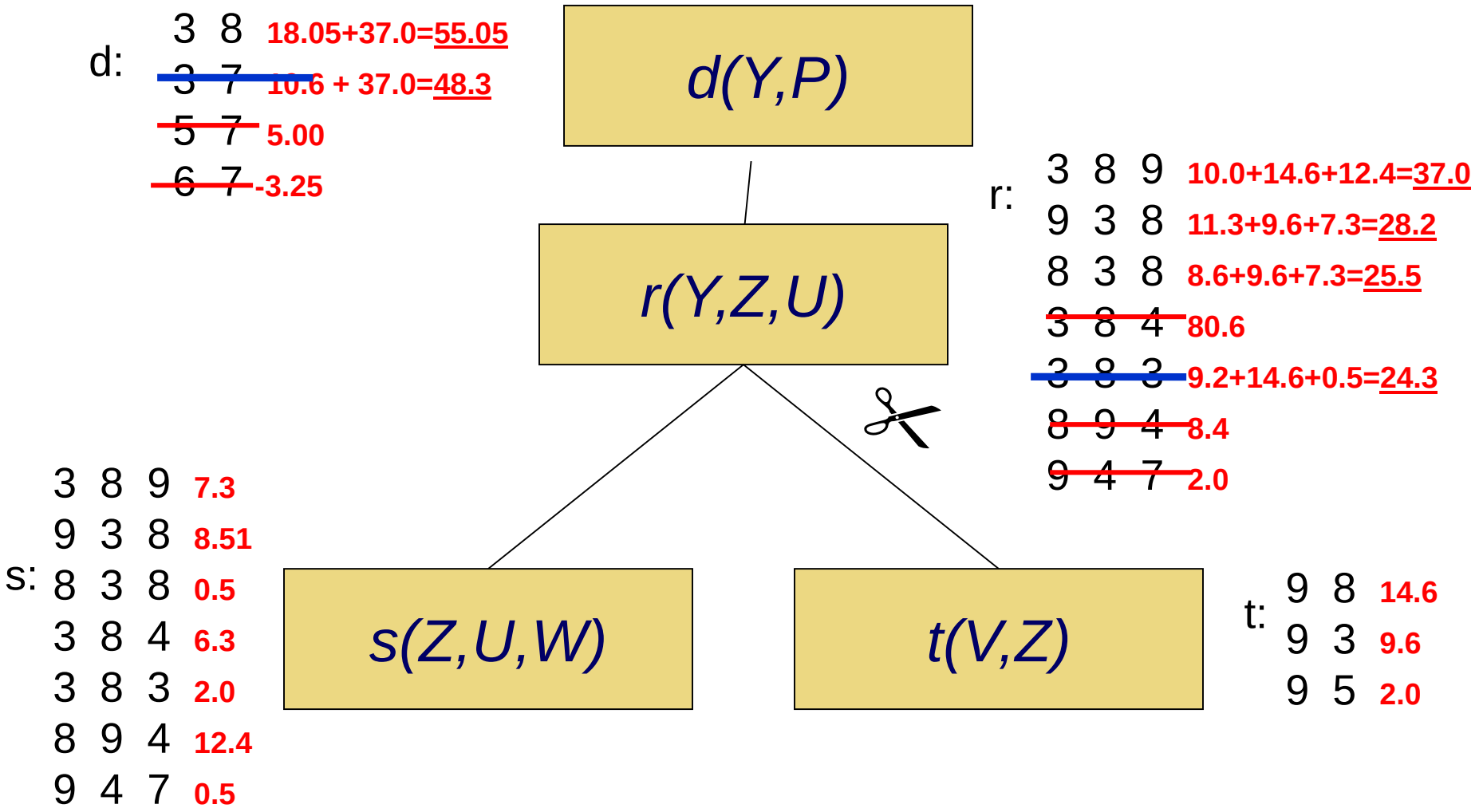
s:

3	8	9	7.3
9	3	8	8.51
8	3	8	0.5
3	8	4	6.3
3	8	3	2.0
8	9	4	12.4
9	4	7	0.5

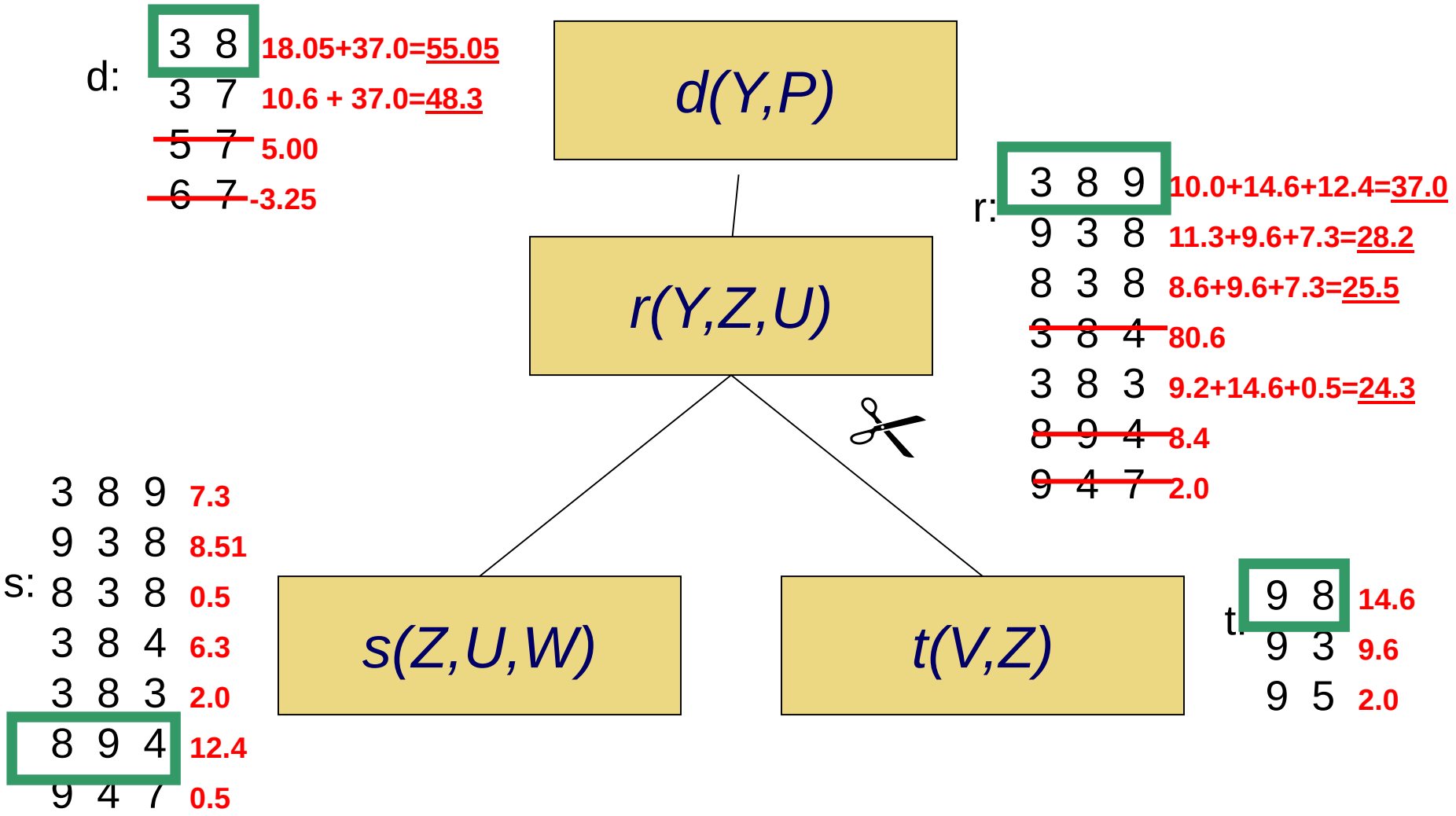
t:

9	8	14.6
9	3	9.6
9	5	2.0

Optimization Version

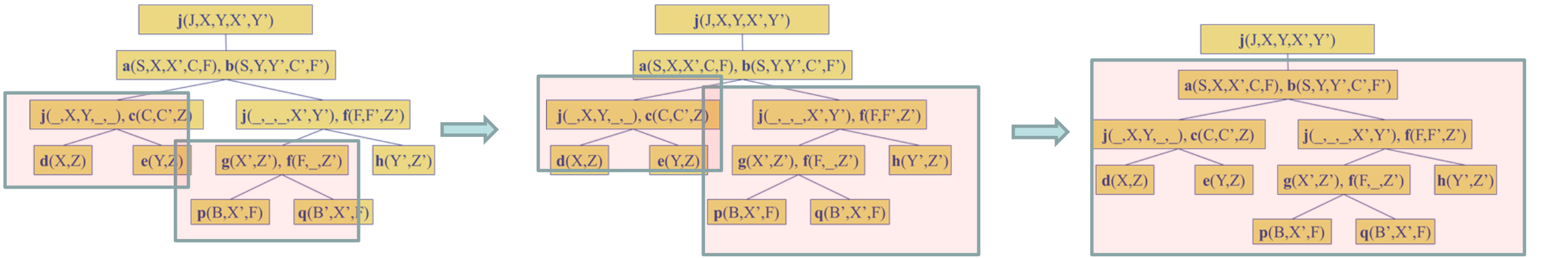


Optimization Version

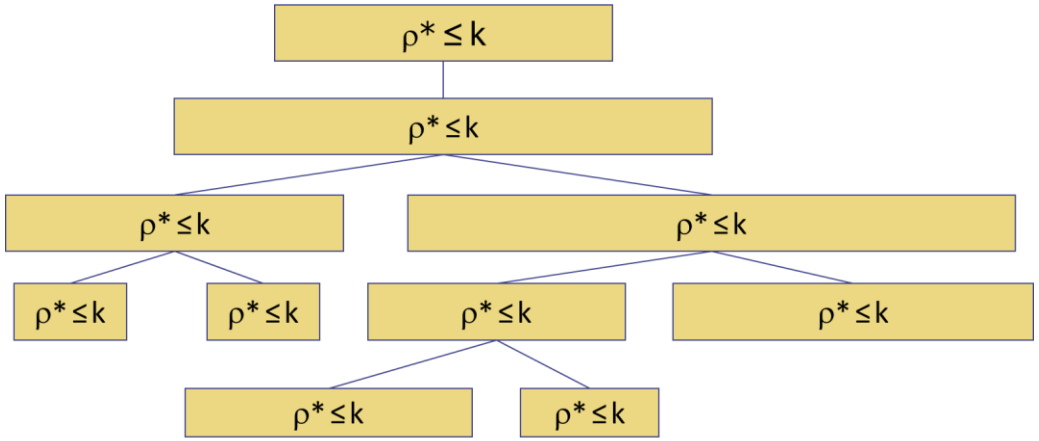


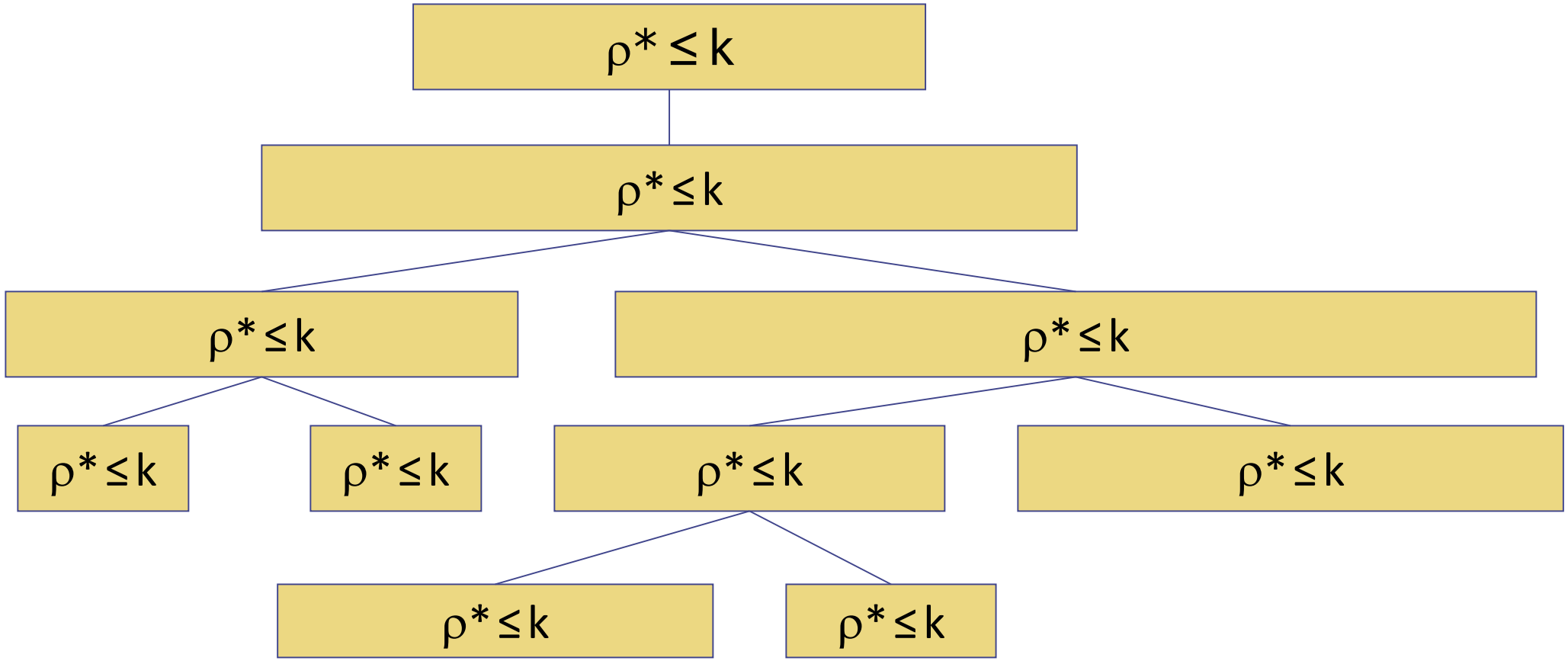
ALOGSPACE algorithm → deterministic polytime algorithms

Opt-k-Decomp: bottom-up construction of HD [G.,Leone,Scarcello 99



Det-k-Decomp (top-down with backtracking & cashing) [G.,Samer


ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k


ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k
ρ* ≤ k