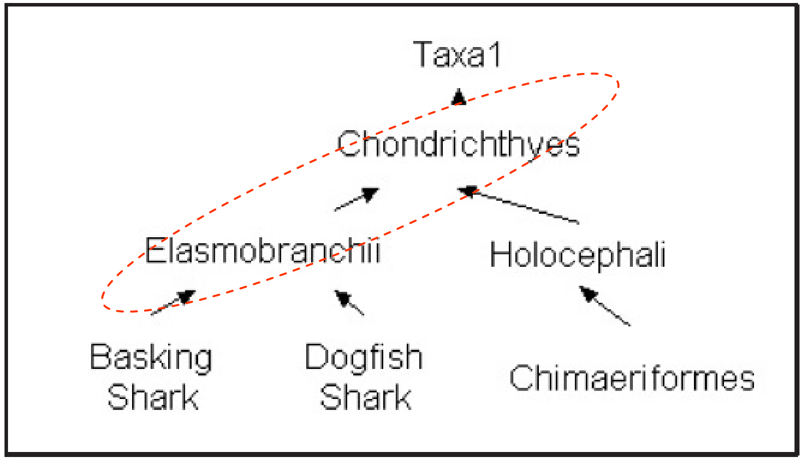
# [HR-Join]

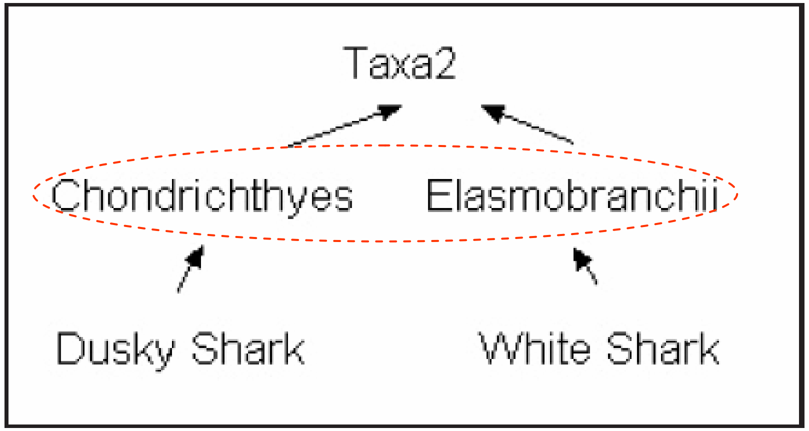
## Sum-Max Monotonic Ranked Joins for Evaluating Top-K Twig Queries on Weighted Data Graphs

| | |
|---|---|
| Yan Qi | Arizona State University |
| K. Selcuk Candan | Arizona State University |
| Maria Luisa Sapino | University of Torino |

# Motivation: Query Processing on Metadata with Conflicts (FICSR[SIGMOD07])



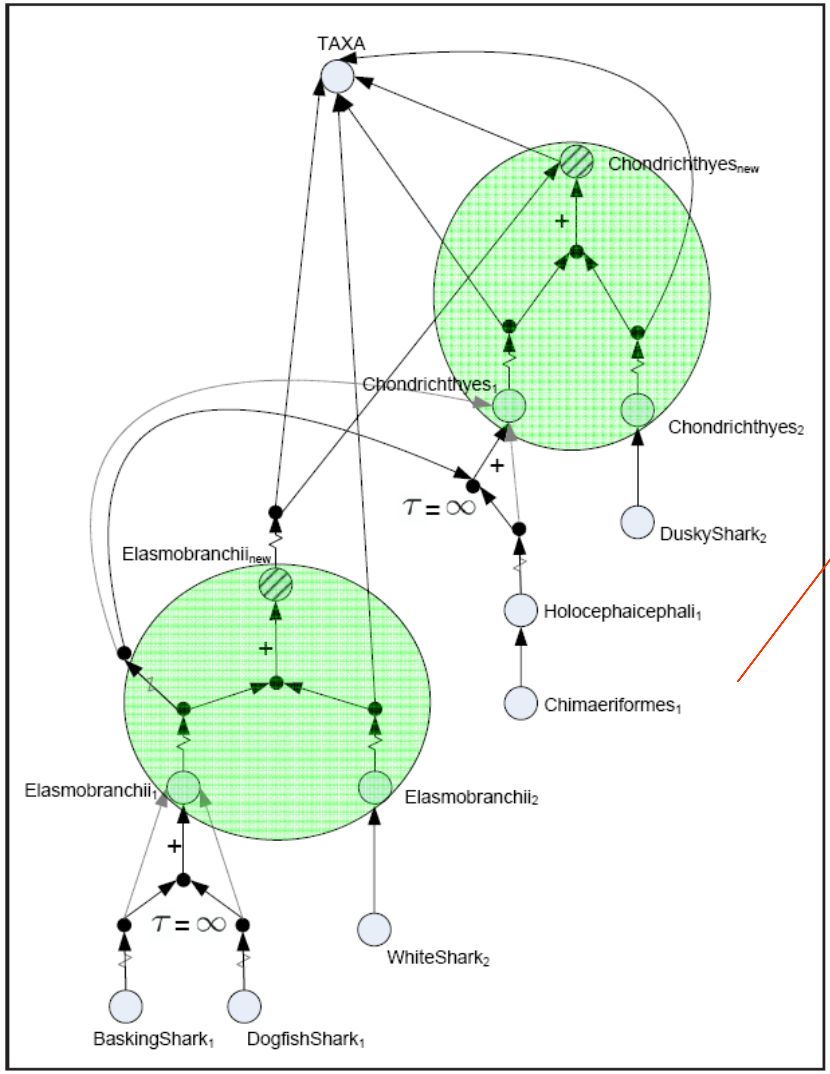
Taxonomy1



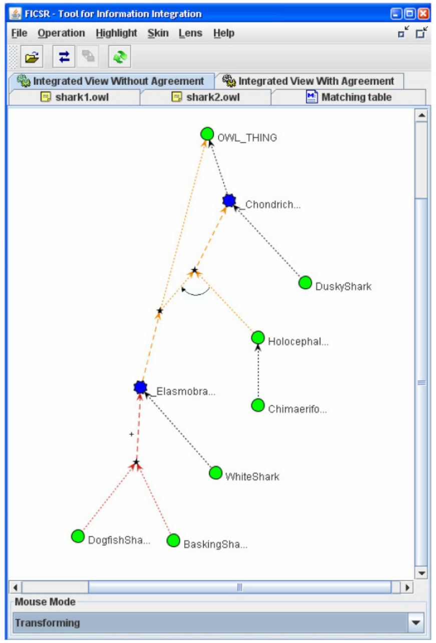
Taxonomy2

Supported by NSF Grant "*Archaeological Data Integration for the Study of Long-Term Human and Social Dynamics (0624341)*"

# FICSR Integrated Representation

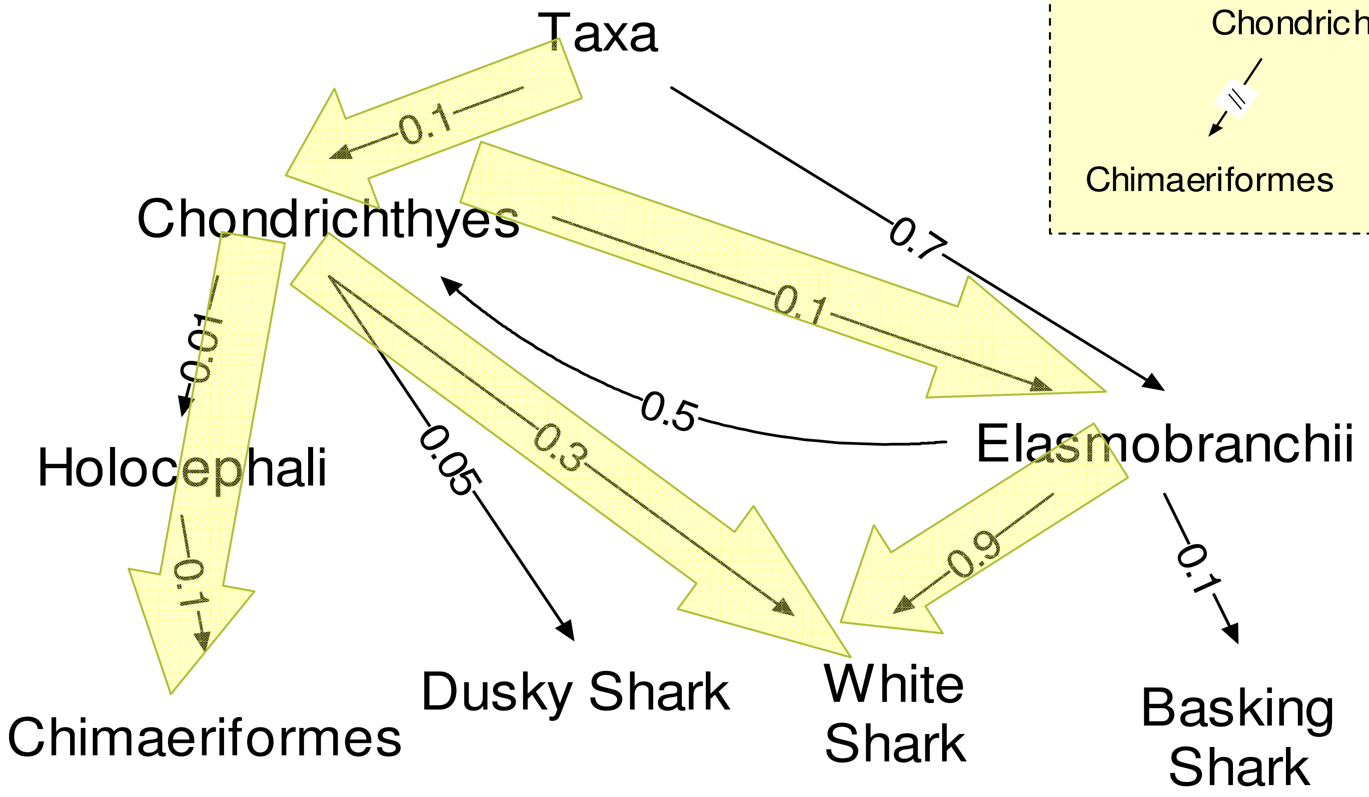


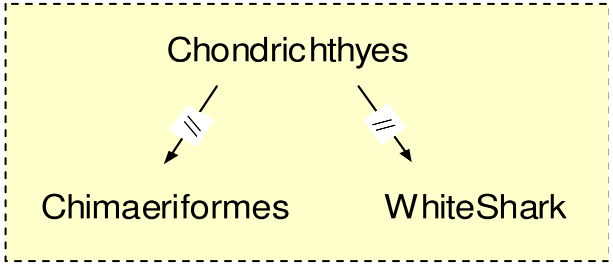
Internal FICSR Representation



Simplified Visualization for the User

# FICSR Agreements [based on source analysis and user feedback]





**Statement of interest:** is it true that “**//Chondrichthyes[//Chimaeriformes]//WhiteShark**“?

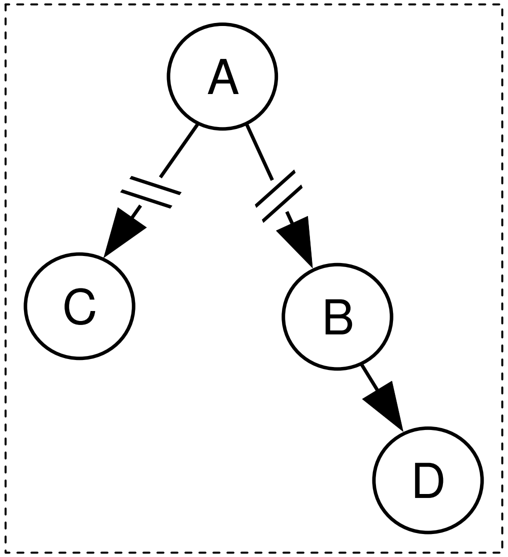
# Related problems

- Web information units [RIU ,Banks, DPBF]
- Keyword search in Relational/OO/XML databases [XRank, ObjectRank, Banks, CP/CV, DPBF]
- Social network analysis [CDIP]

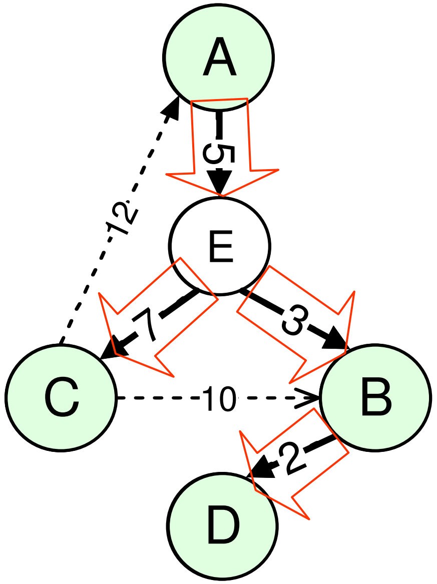
Common theme:

- Data is a graph…
- …relevant content is distributed across the graph…
- …but, queries (e.g. keyword sets) are not structured.

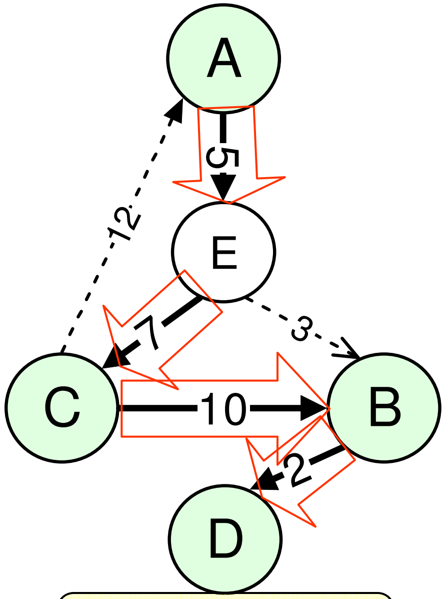
# Twig queries (i.e., structure of interest) and top-k results



A[//C]//B/D



Cost = 17

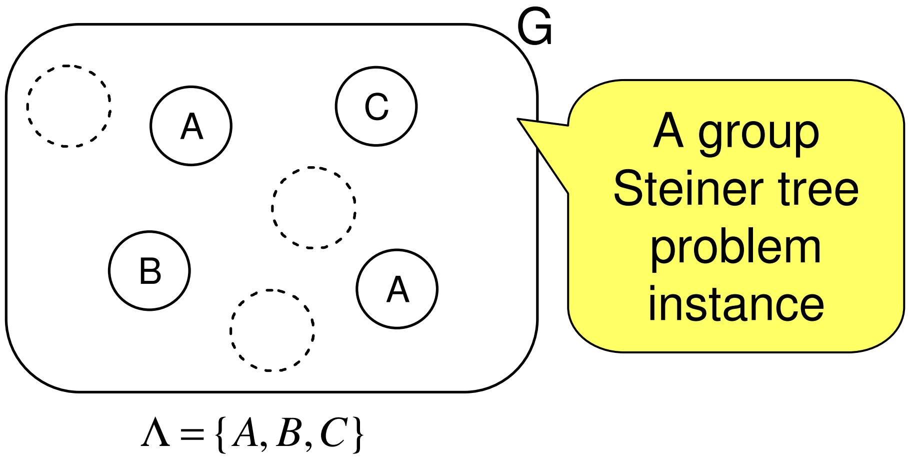


Cost = 24

More desirable….can we find it before the other??

# Answering twig queries on weighted graphs

- So, how hard is the "min-cost twig query problem"?
- NP-complete (by reduction from the "*group Steiner tree* problem")

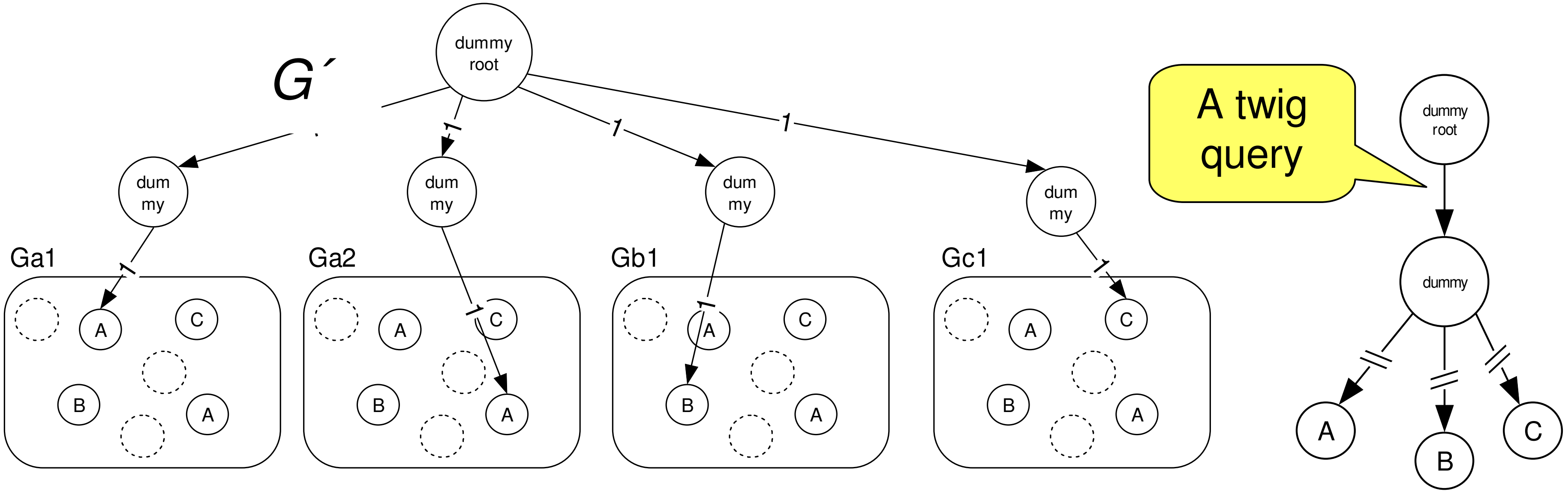


$\Lambda = \{A, B, C\}$

A group Steiner tree problem instance

# Answering twig queries on weighted graphs

- So, how hard is the "min-cost twig query problem"?
- NP-complete (by reduction from the "*group Steiner tree* problem")

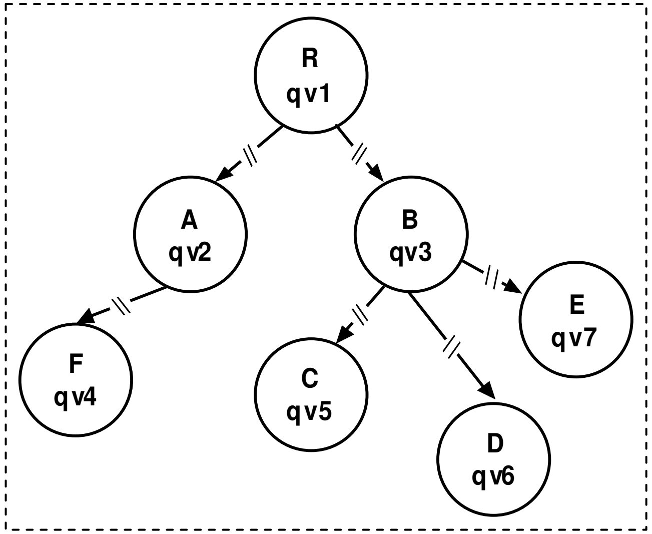


A twig query

Also see DBTwig results (Kimelfeld and Sagiv, 06)

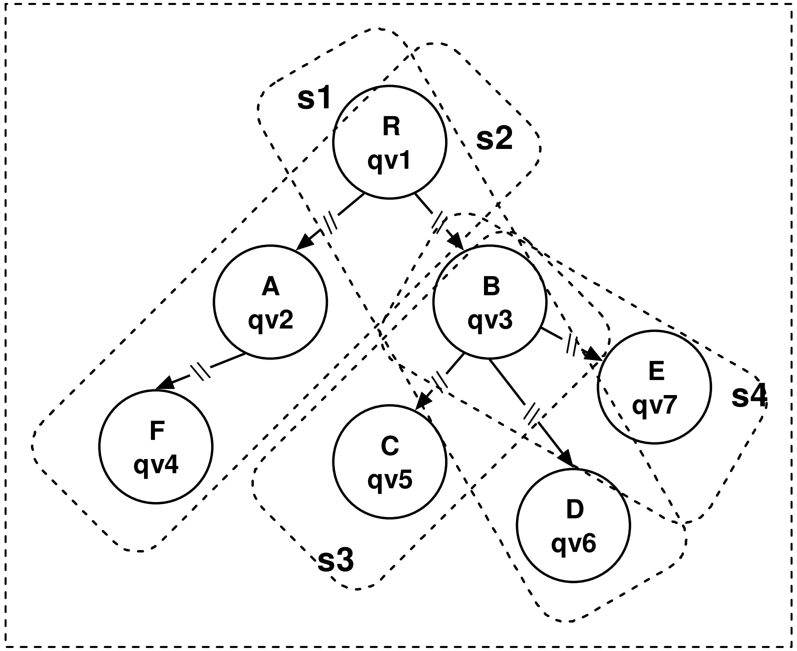
# So what can we do?

- Keyword search on graph data [RIU, BANKS] ?
  - No…we need to enforce query structure..



# So what can we do?

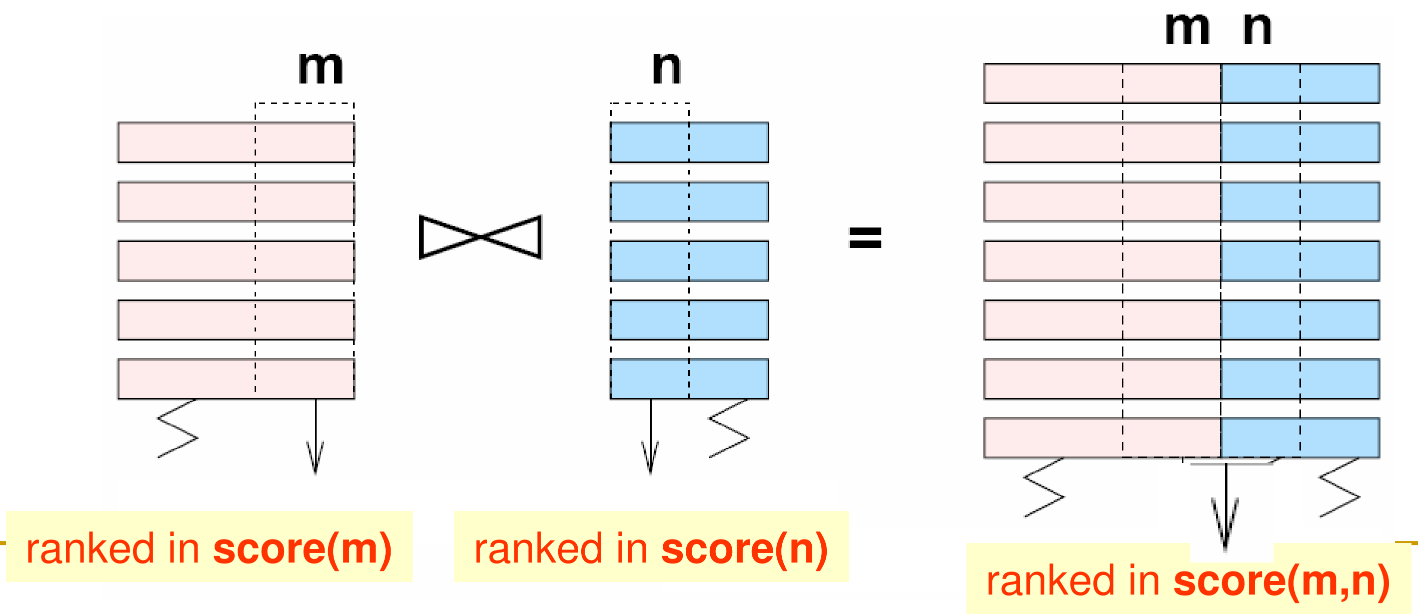
- Keyword search on graph data [RIU, BANKS] ?
  - No…we need to enforce query structure..
- Ranked-join algorithms (FA,TA, NRA) for top-k queries



$$Q = s_1 \rhd\lhd s_2 \rhd\lhd s_3 \rhd\lhd s_4$$

# So what can we do?

- Keyword search on graph data [RIU, BANKS] ?
  - No…we need to enforce query structure..
- Ranked-join algorithms (FA,TA, NRA) for top-k queries
  - ….score combination function must be monotonic.



ranked in **score(m)**

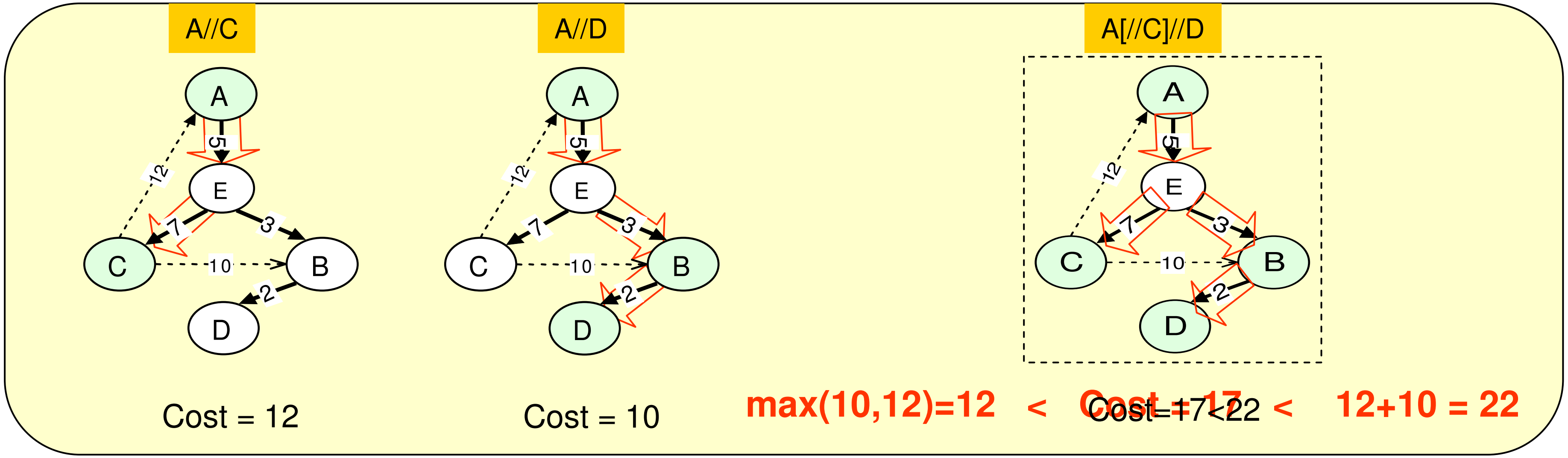
ranked in **score(n)**

ranked in **score(m,n)**

# Sum-Max Monotonicity

- Ranked joins is a good idea…
  - ..but, monotonicity does not hold.
- Good news: Sum-Max monotonicity

$$\max_{sr_i \in R}(\text{cost}(sr_i)) \leq \text{cost}(R) \leq \sum_{sr_i \in R} \text{cost}(sr_i)$$



A//C

Cost = 12

A//D

Cost = 10

A[//C]//D

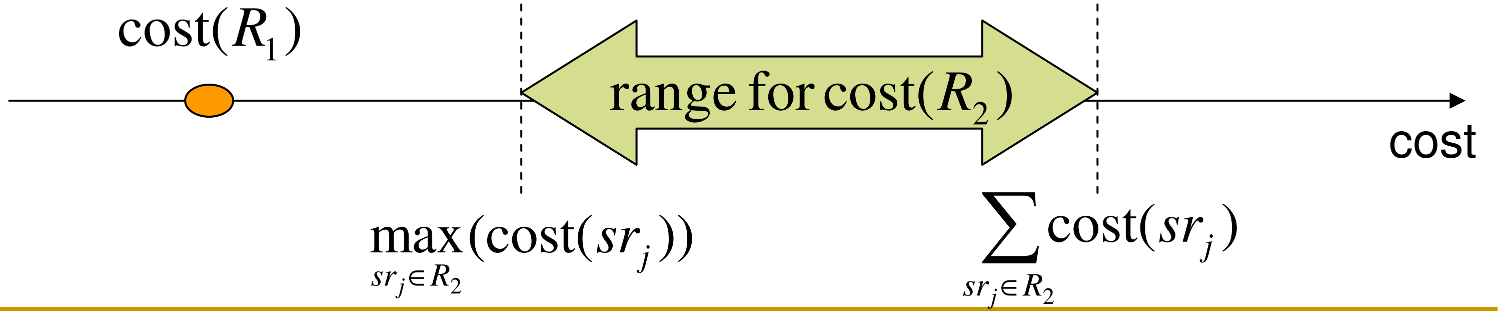
Cost=17<22

max(10,12)=12 < Cost=17 < 12+10 = 22

# Sum-Max Monotonicity

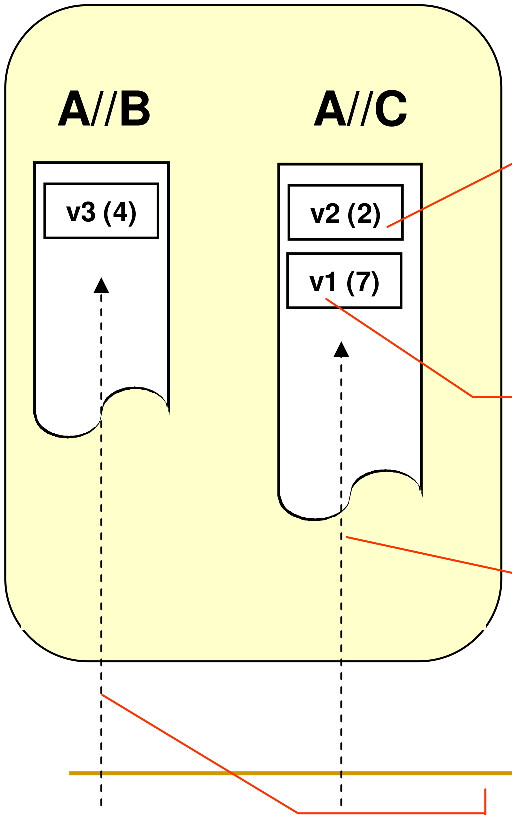
- Ranked joins is a good idea…
  - ..but, monotonicity does not hold.
- In fact, we can also see that

$$\left(\mathrm{cost}(R_1) \le \max_{sr_j \in R_2}(\mathrm{cost}(sr_j))\right) \rightarrow \mathrm{cost}(R_1) \le \mathrm{cost}(R_2)$$


$\mathrm{cost}(R_1)$
range for $\mathrm{cost}(R_2)$
cost
$\max_{sr_j \in R_2}(\mathrm{cost}(sr_j))$
$\sum_{sr_j \in R_2} \mathrm{cost}(sr_j)$

# Progressive enumeration based on Sum-Max monotonicity

- Query: A[//B]//C



matches are arriving in increasing cost order

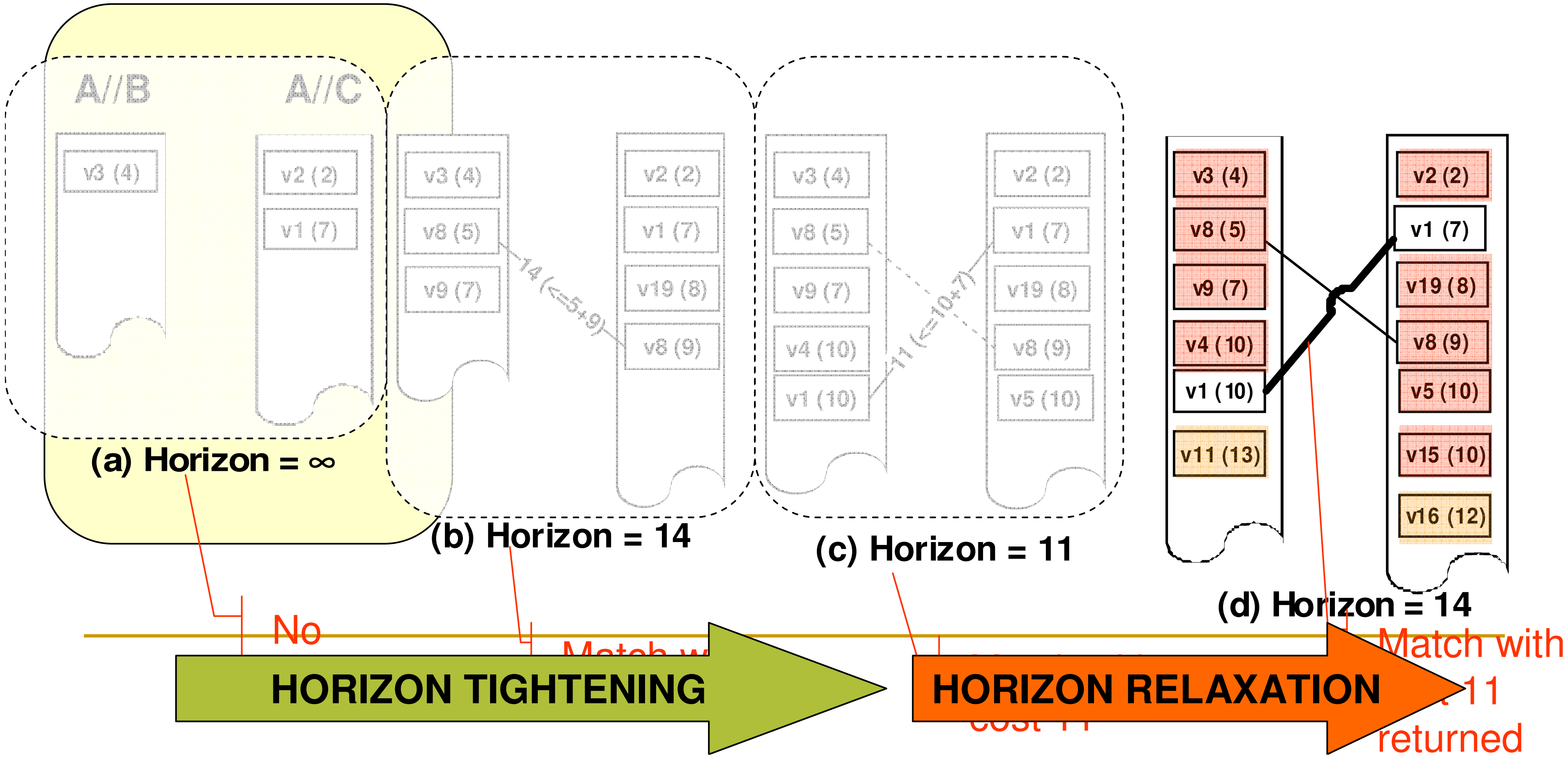
**vi**'s denote the join nodes (with label "A")

Stream of matches to A//C

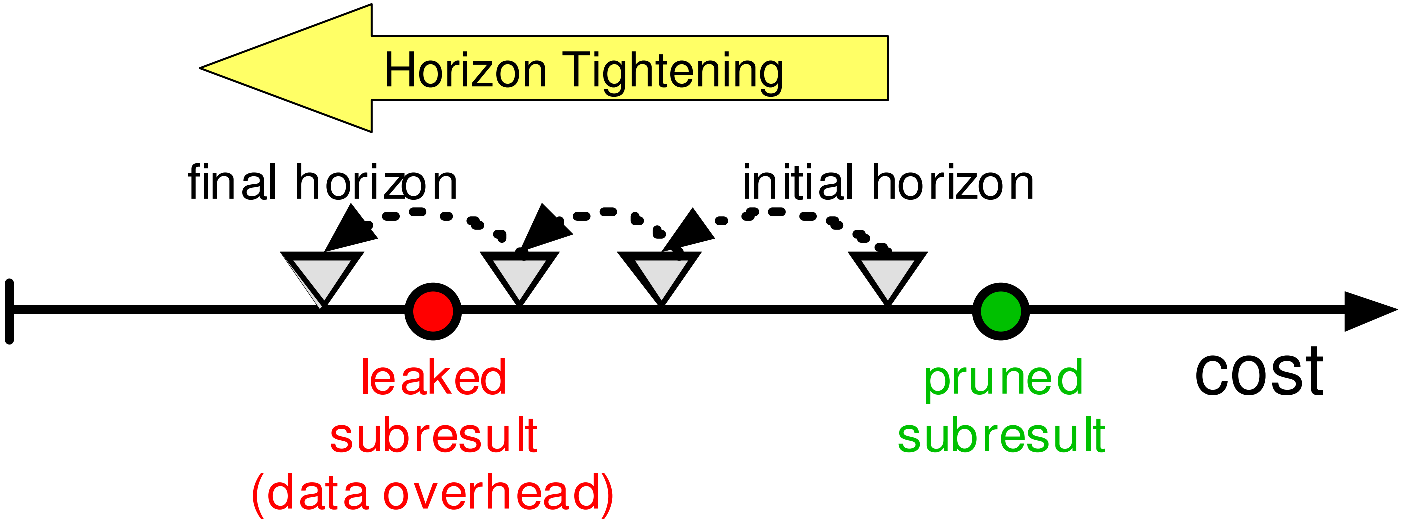
Stream of matches to A//B

# Progressive enumeration based on Sum-Max monotonicity

- Horizon -> Stopping criterion

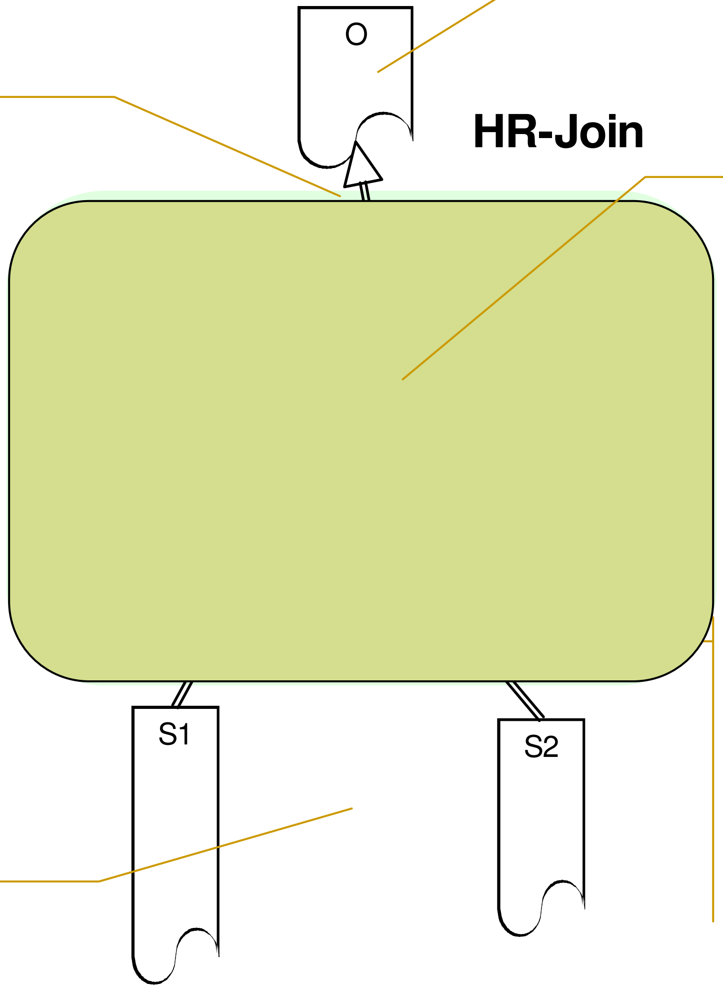


# Pruning and data overhead



$$dataoverhead = \frac{\#all\_submatches - \#necessary\_submatches}{\#all\_submatches}$$

# HR-Join: Horizon based Ranked Join



Ordered output stream

Result sieve controls when a candidate match be declared “result”

Implements sum-max pruning

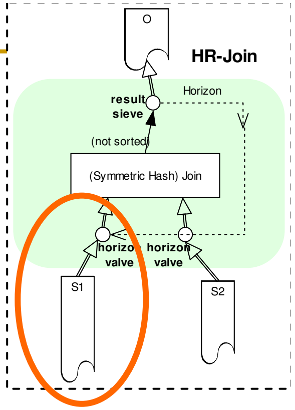
**Relies on the “punctuations” received from the horizon valves**

Horizon valve regulates the availability of the incoming data to the hash join.

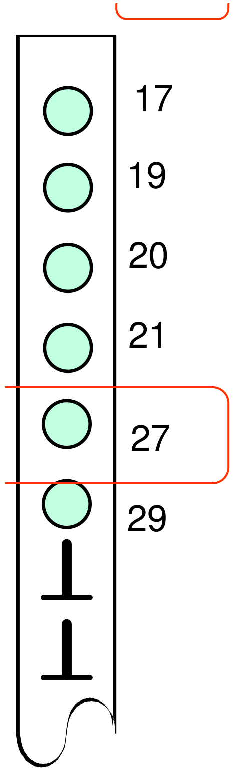
**Relies on the “horizon” value from result sieve**

Two ordered streams

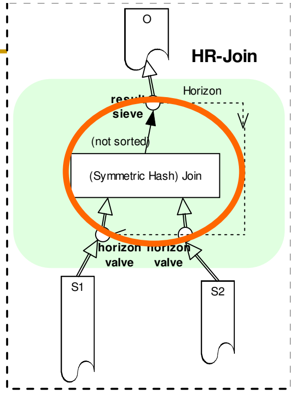
# Operation of the horizon valve



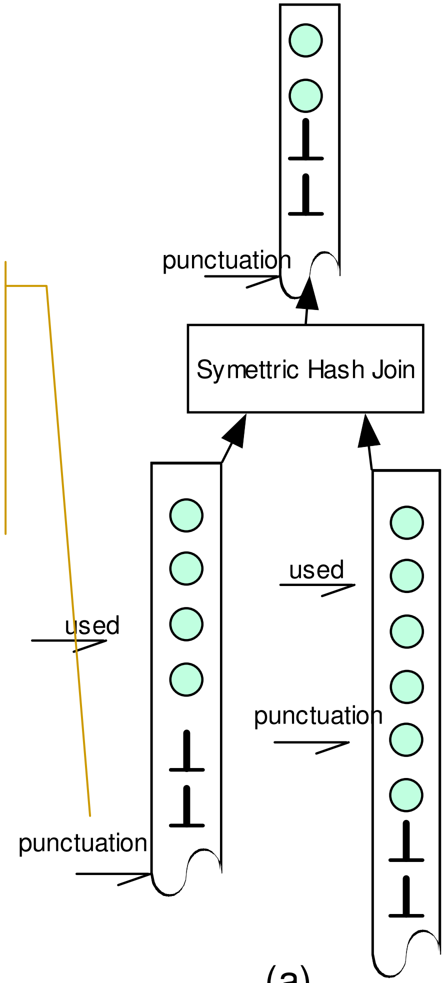
- Punctuation: blocking due to the horizon limit



# Punctuations are propagated by the symmetric-hash join



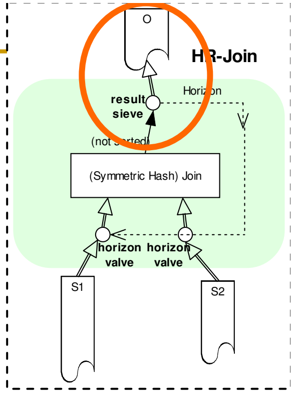
Punctuations in the input streams are not met yet

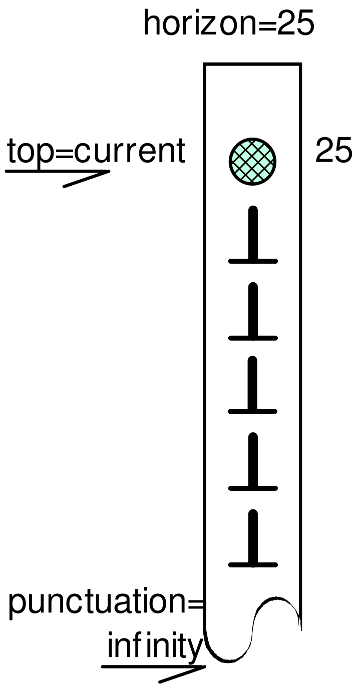


(a)

# Operation of the result sieve

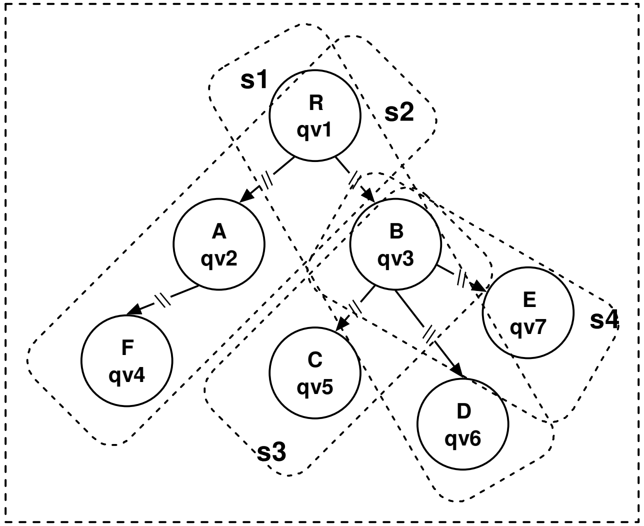
- Top: The current best
- Punctuation: indicates that the input streams are punctuated

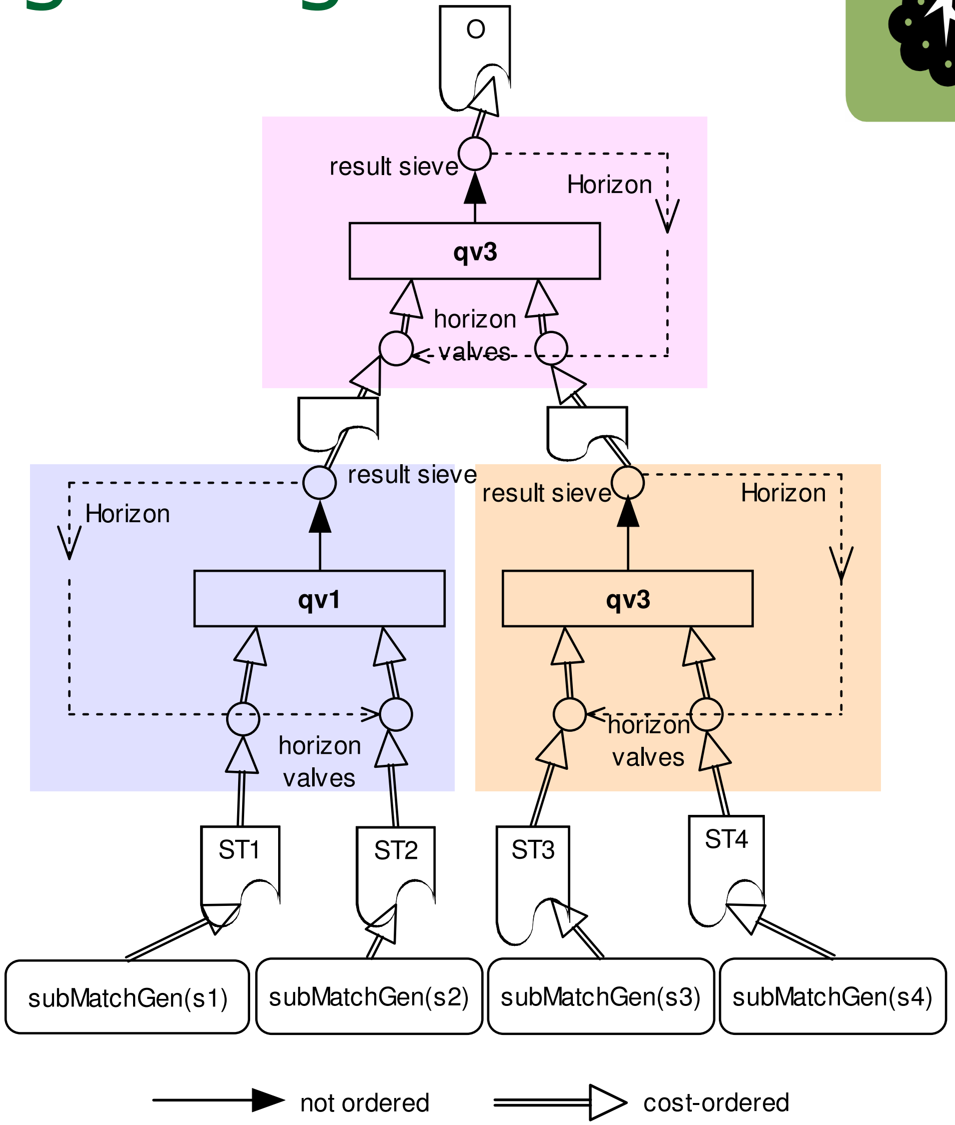




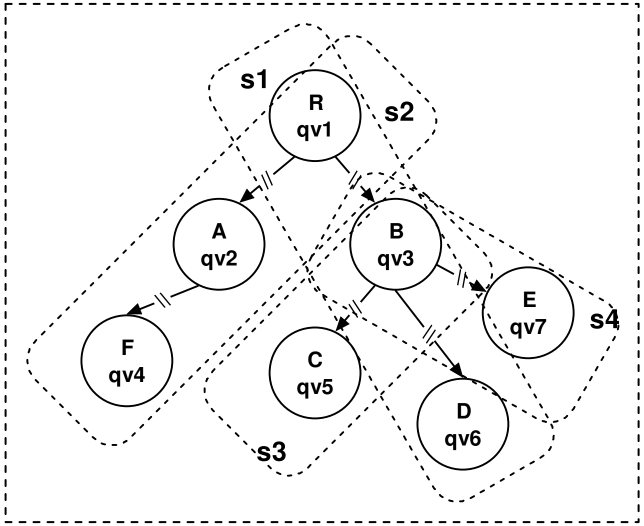
Indirectly regulates its own input stream by updating the horizon value

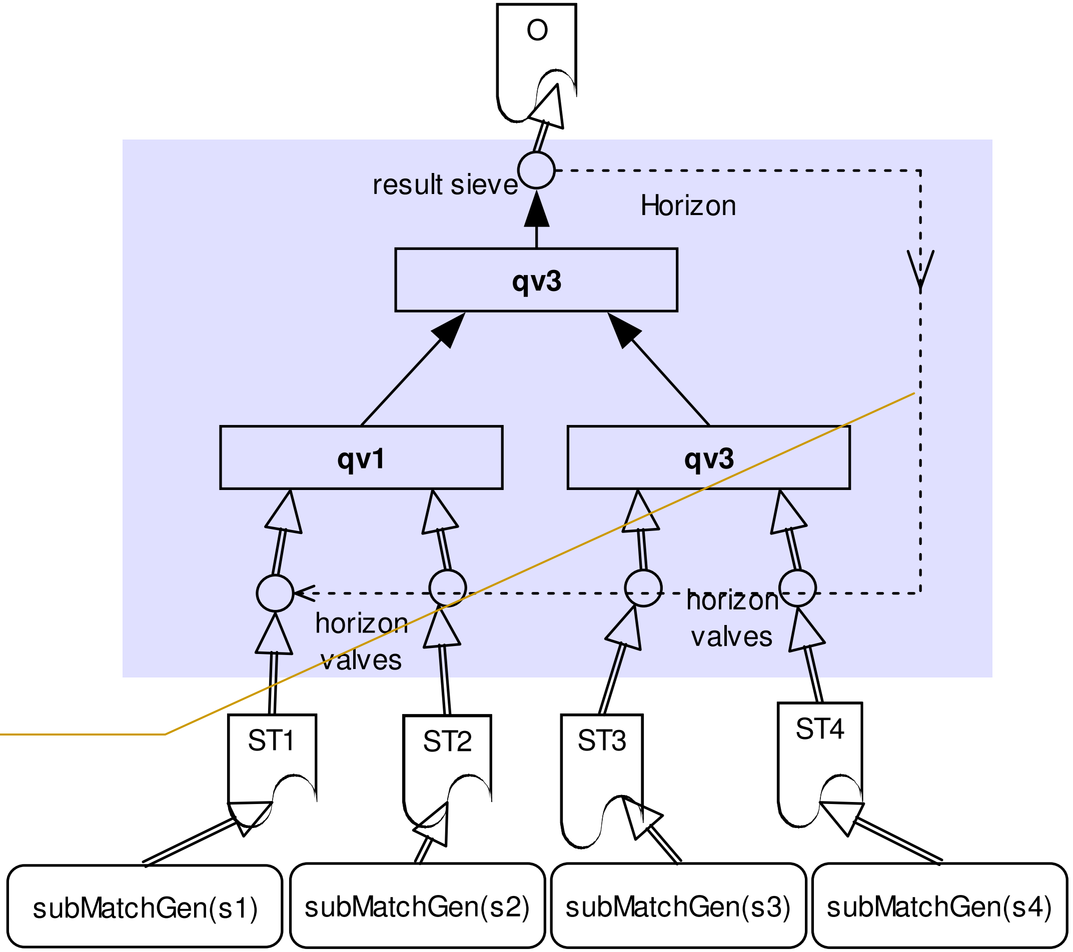
# Query processing using HR-Join





# M-way HR-Joins





Only one result sieve regulating all valves of the four input streams.

# What is missing?

- How to enumerate (subresult) paths in cost order?
  - K-shortest simple paths problem [Qi et al. SIGMOD 07]

    $$O(k|V|(|E|+|V|\log|V|))$$

    - ..details are in the paper

- How to deal with “*” wildcards in twigs??
  - can be expensive (too many matches and joins)
  - query rewriting…....details are in the paper

# Can we do better?

- Horizon values are set based on

$$\left(\mathrm{cost}(R_1) \leq \max_{sr_j \in R_2}(\mathrm{cost}(sr_j))\right) \rightarrow \mathrm{cost}(R_1) \leq \mathrm{cost}(R_2)$$

which assumes the worst case:

i.e., subresults may overlap fully.

- Horizon tightening factor (tf) can be used when overlaps are known to be bounded

$$\left(\mathrm{cost}(R_1) \leq \boxed{tf} \max_{sr_j \in R_2}(\mathrm{cost}(sr_j))\right) \rightarrow \mathrm{cost}(R_1) \leq \mathrm{cost}(R_2)$$

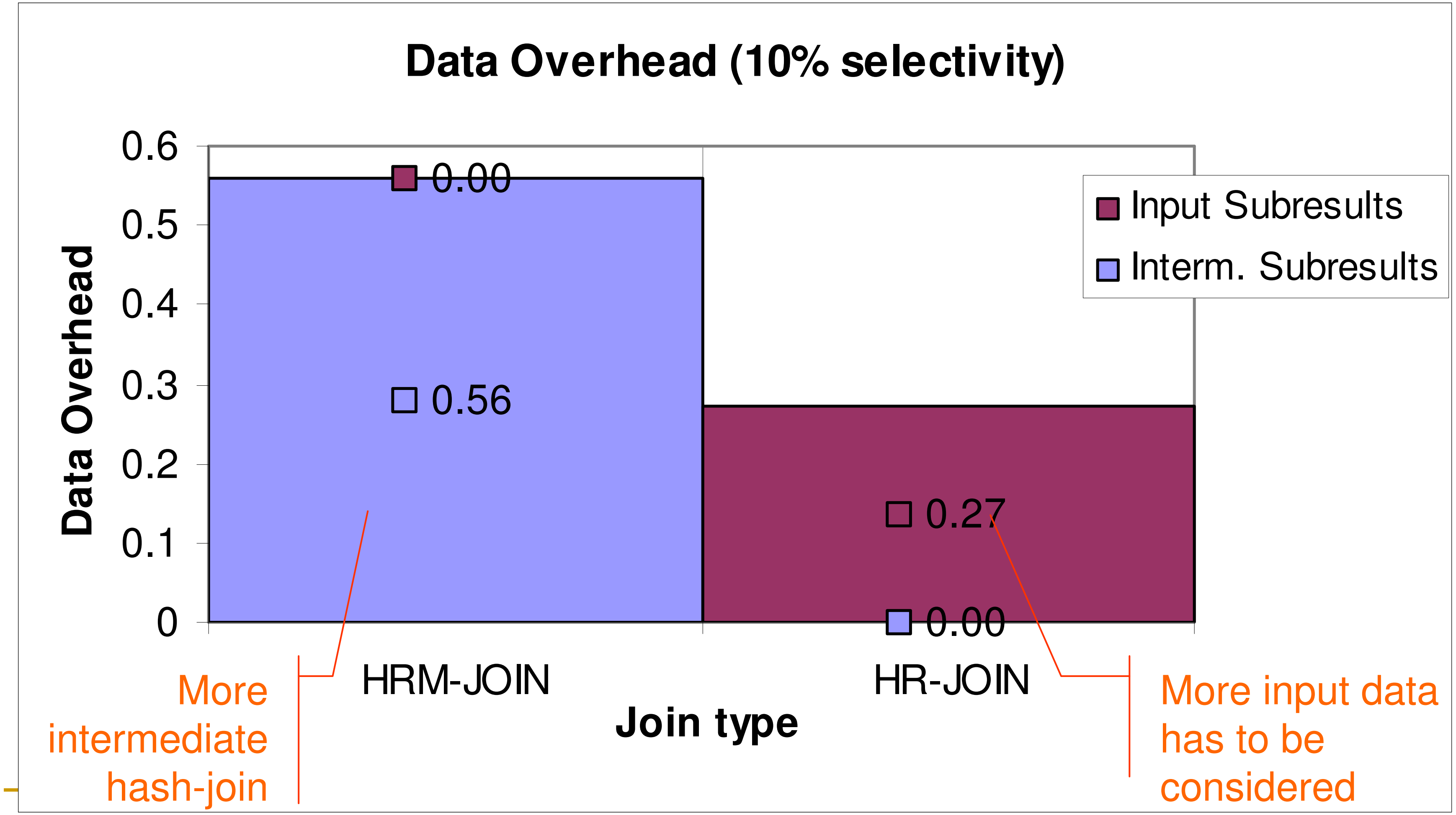
# Experiments

- 2GHz Pentium with 1GB main memory.

- Query plans:
  - HR-Join and M-Way HR-Join (MHR-Join)
  - 2 significantly different join-selectivity distributions: ~10% and 1-to-1.

- Data
  - FICSR weighted graph data
    - Zipfian-like distribution of edge weights

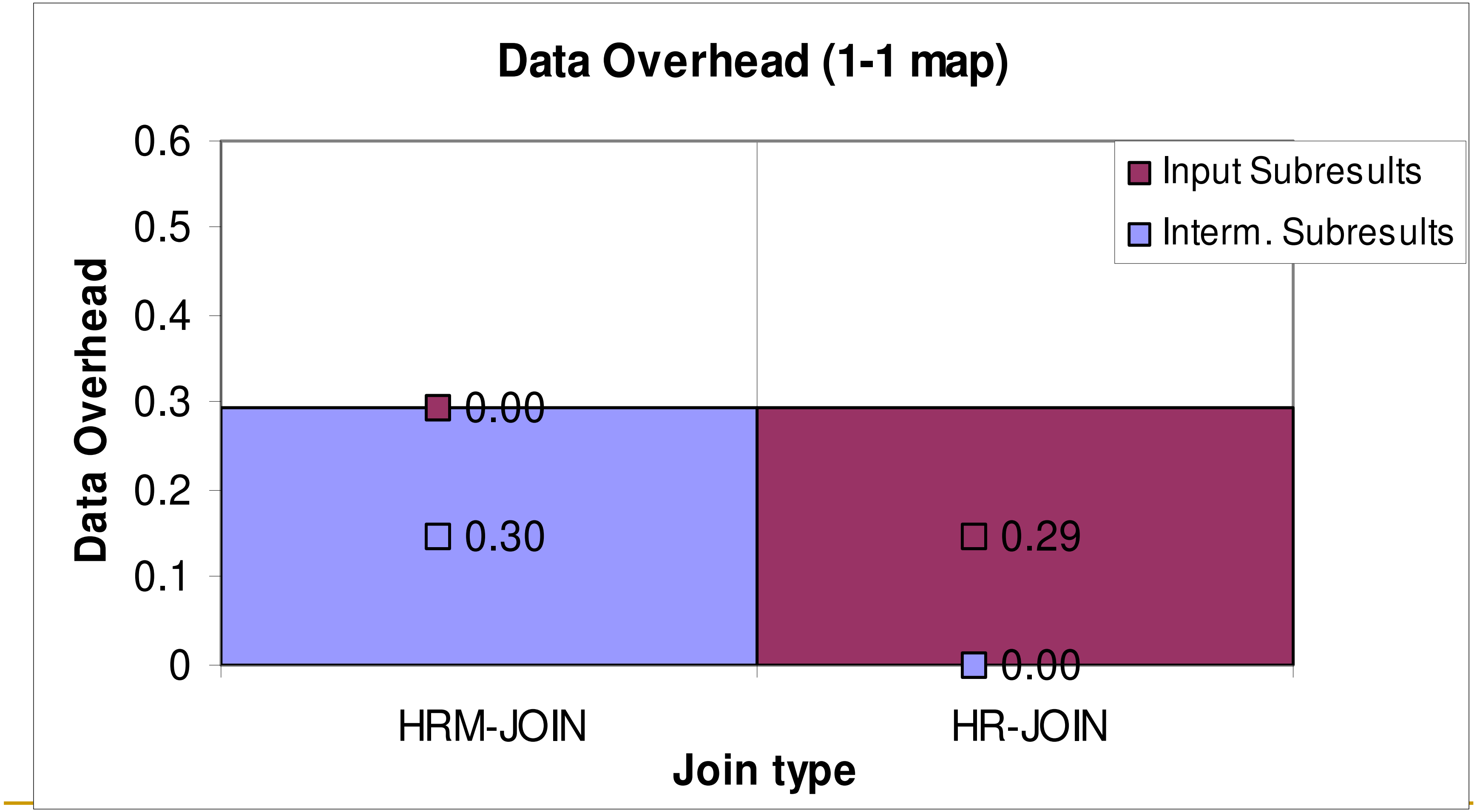
# Data overhead of HRM-Join versus HR-Join



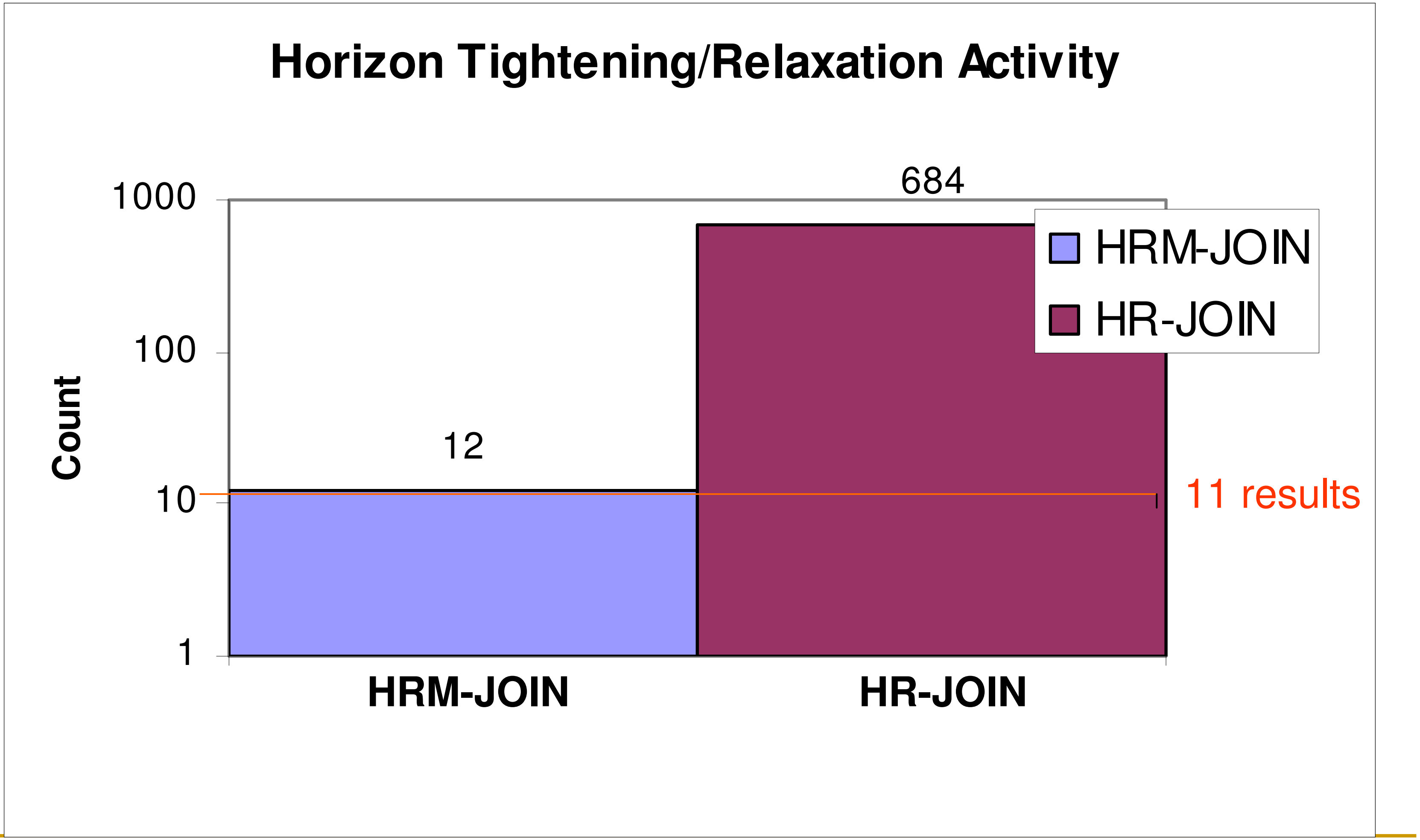
More intermediate hash-join results

More input data has to be considered

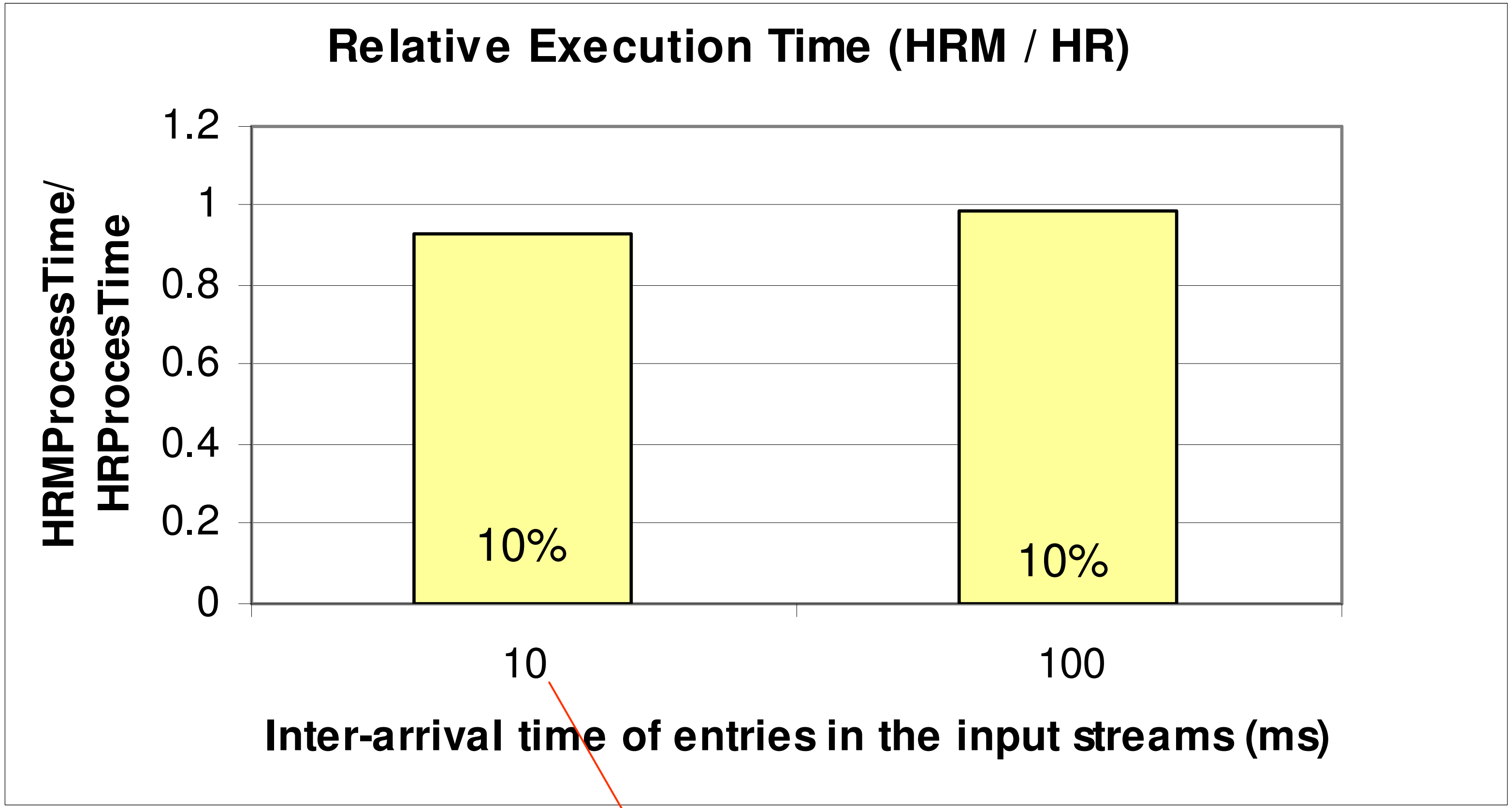
# Data overhead of HRM-Join versus HR-Join



# HR-Join has higher horizon-management cost (10%)

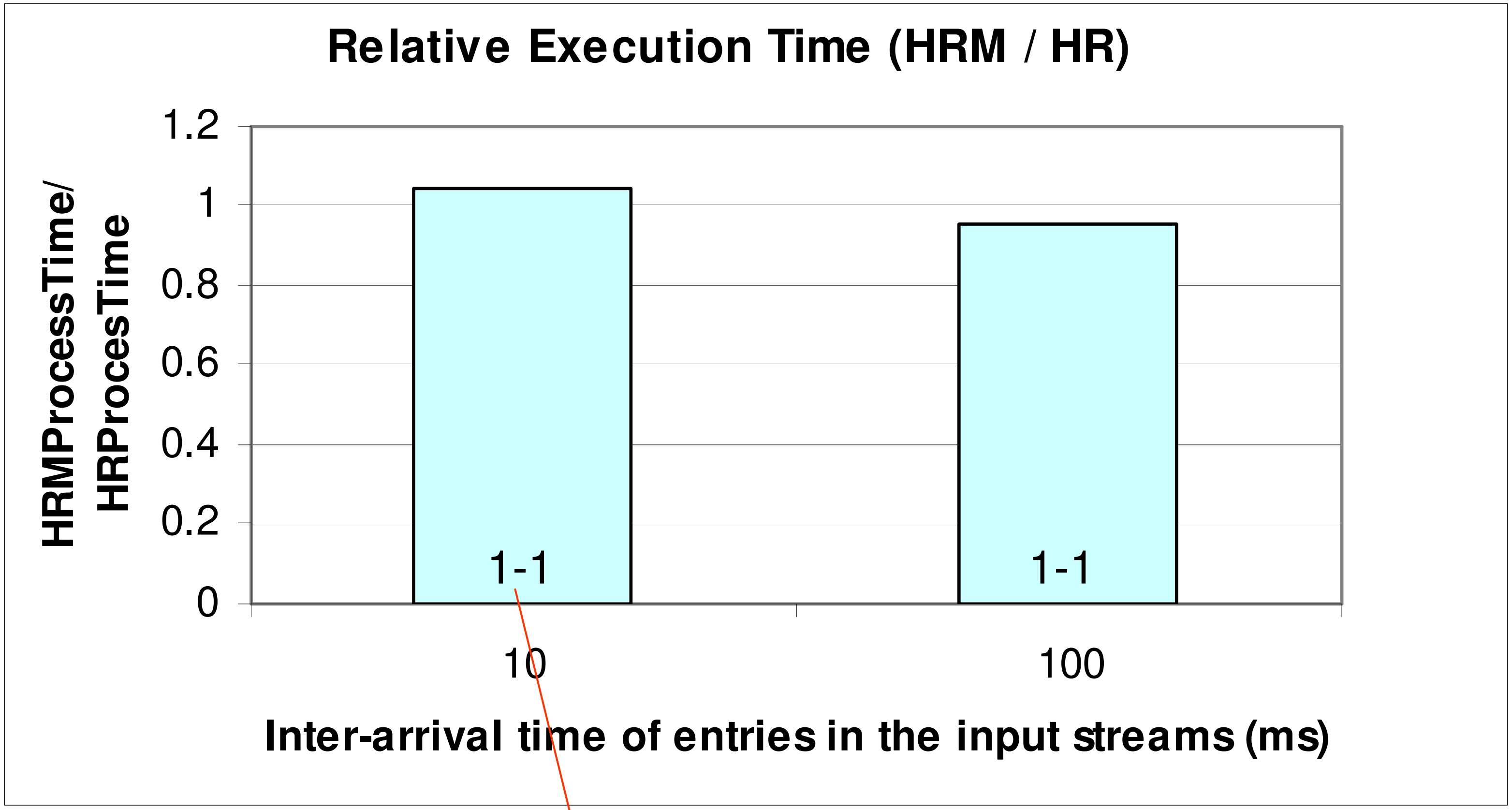


# Inter-arrival time of stream inputs



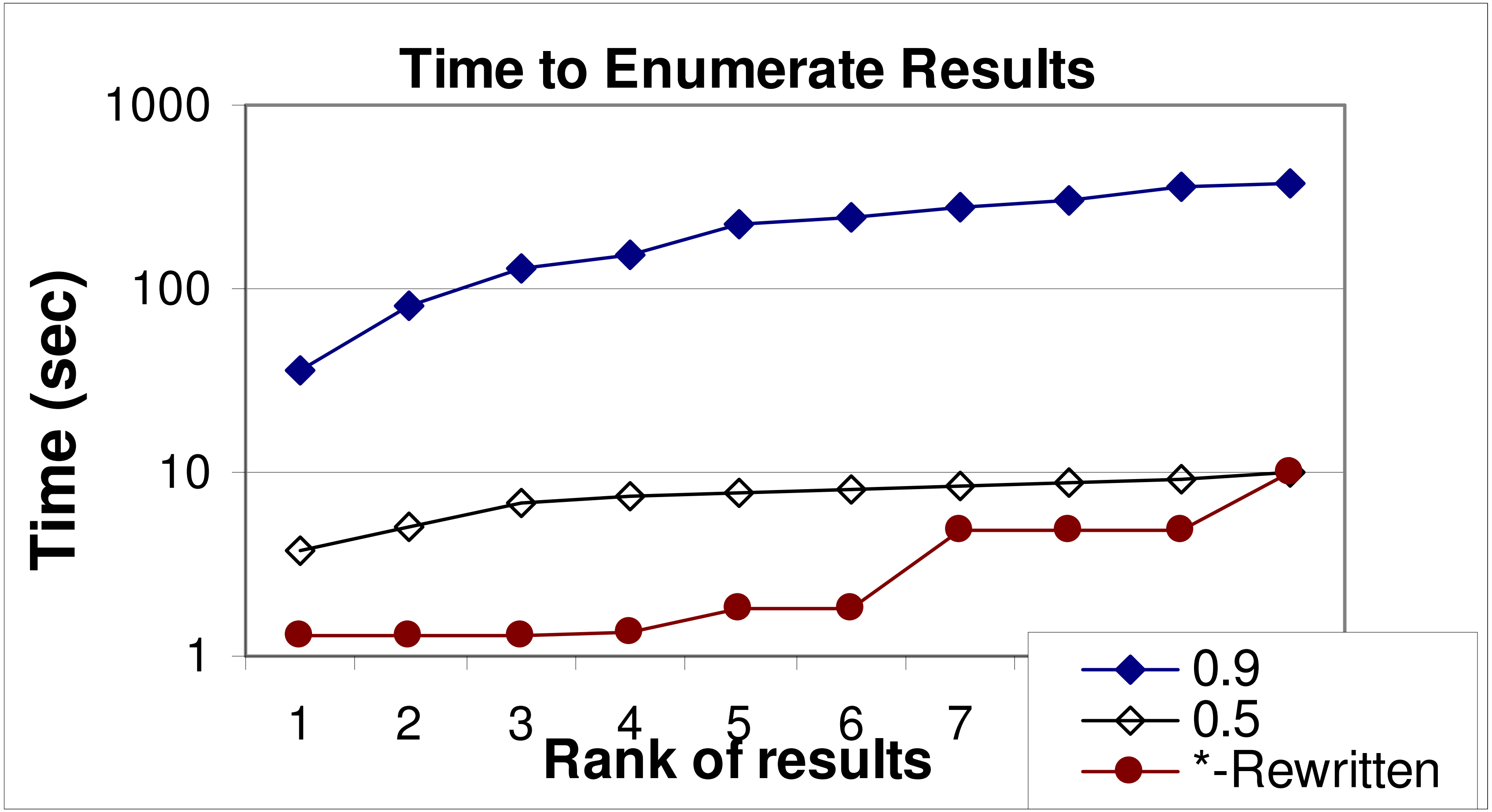
When inter-arrival is tight, HRM benefits from reduced horizon management

# Inter-arrival time of stream inputs



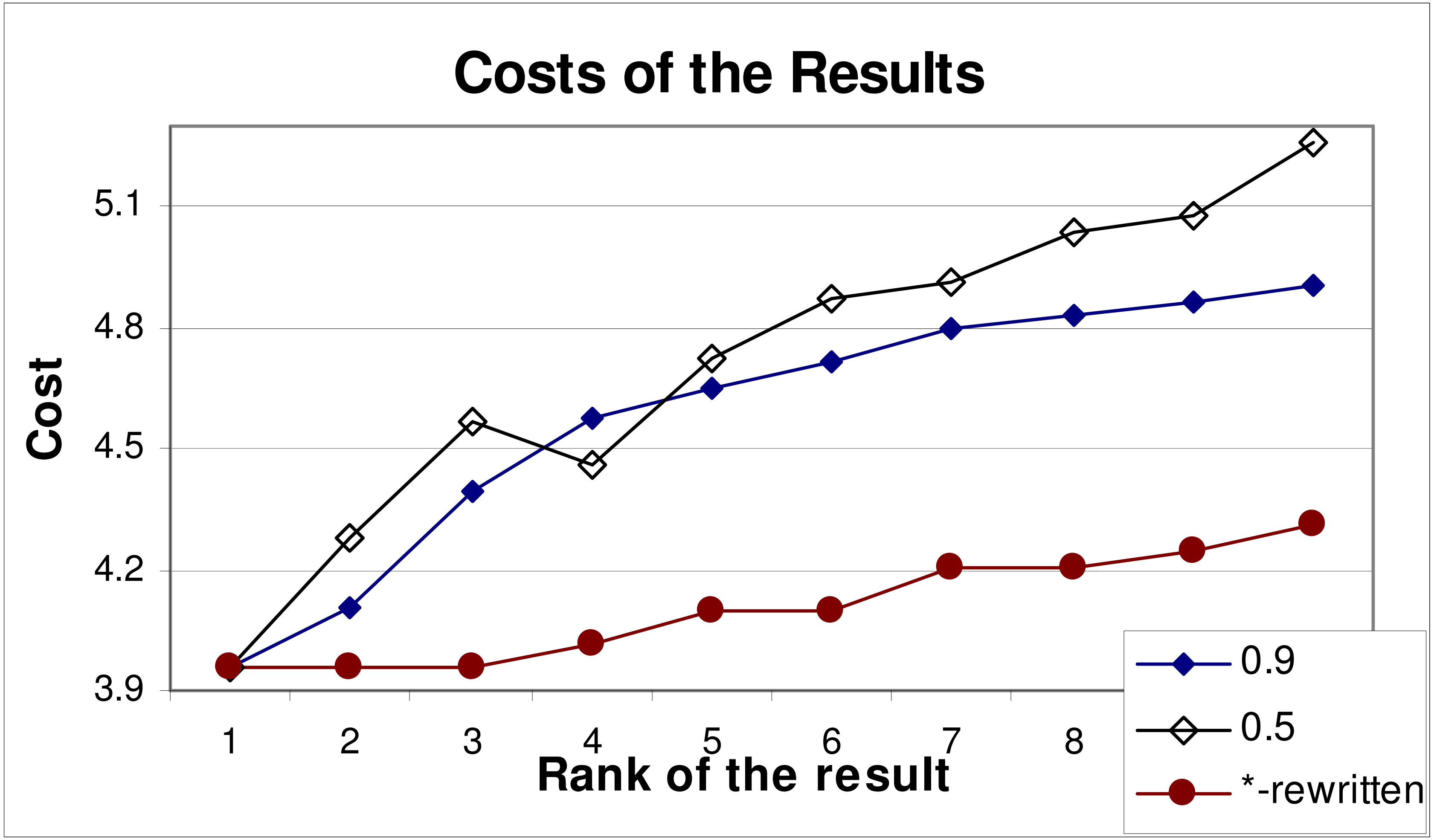
When selectivity is tight, there is less horizon tightening, thus HR-Join performs OK

# Horizon tightening and *-rewriting help with "wildcard" queries



Query: A//*[//B]//C

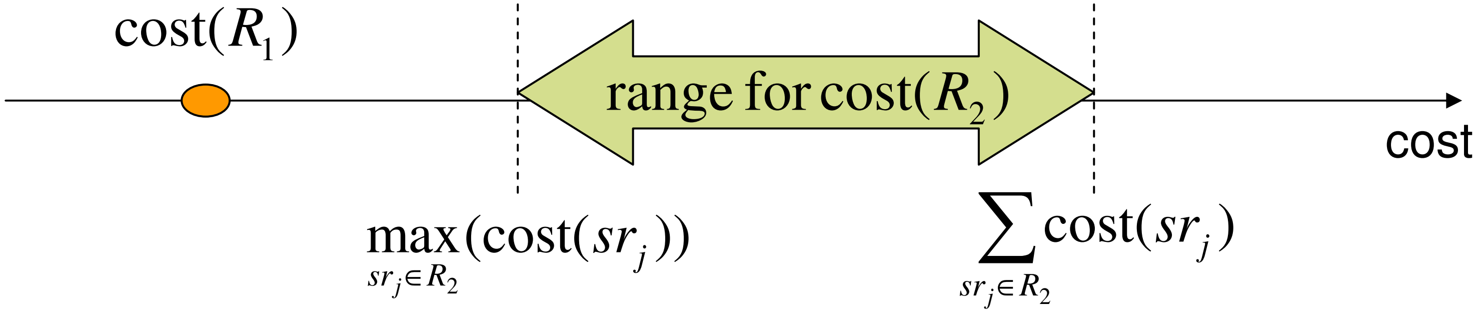
# The costs of the distinct results of the rewritten query are significantly better



Query: A//*[//B]//C

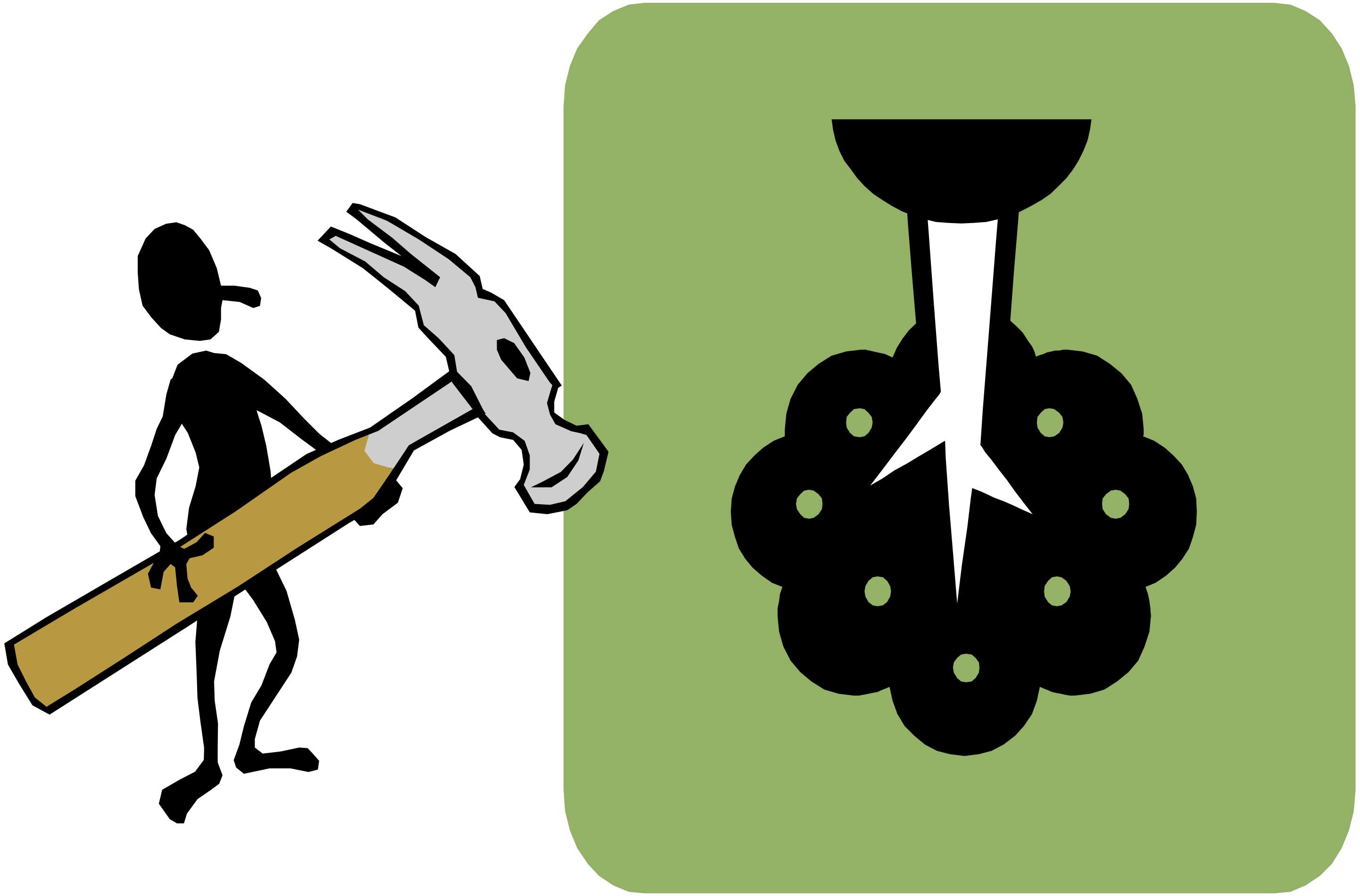
# Conclusion

- Sum-max monotonicity…

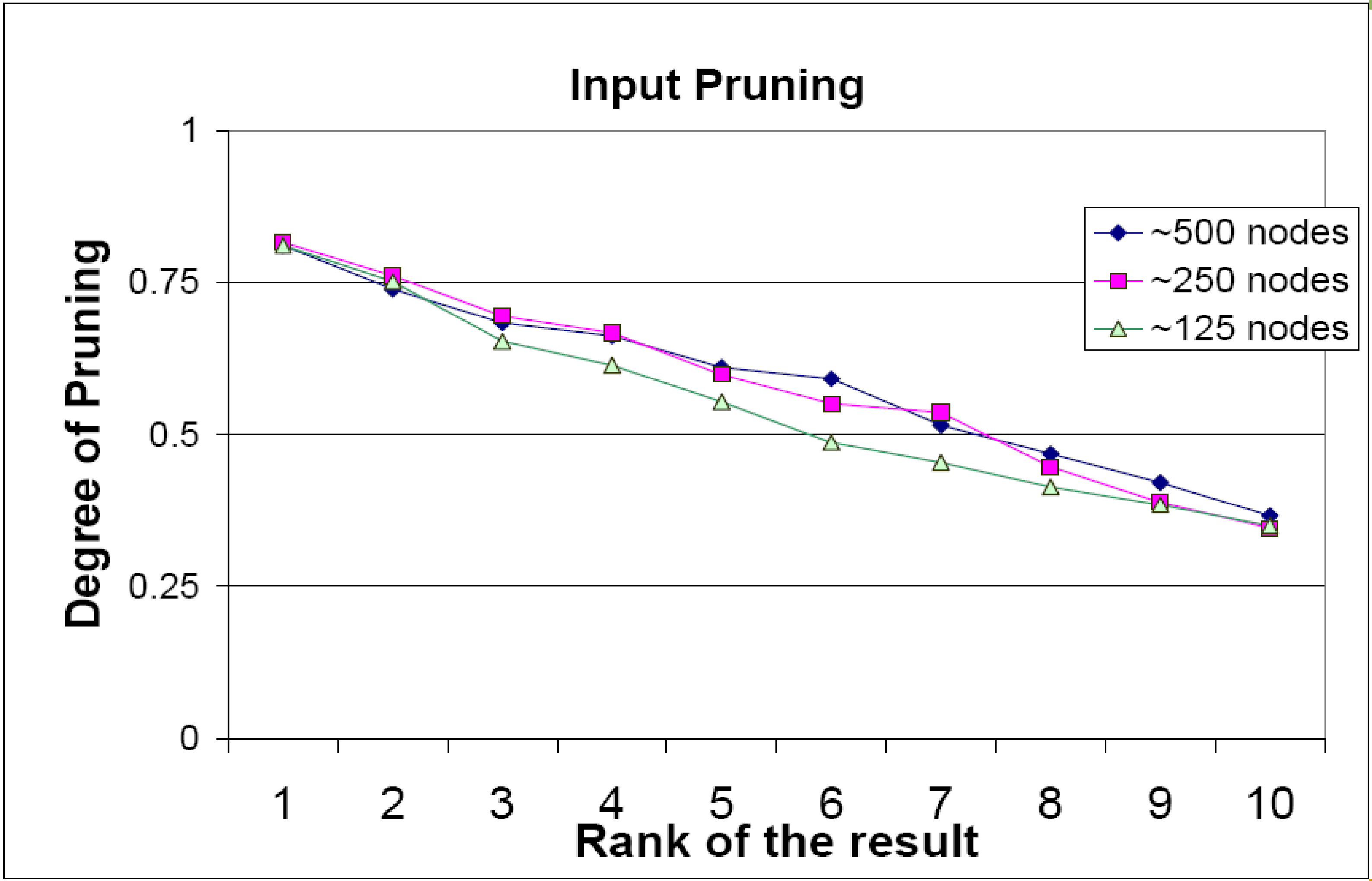

$\text{cost}(R_1)$
$\text{range for cost}(R_2)$
cost
$\max_{sr_j \in R_2}(\text{cost}(sr_j))$
$\sum_{sr_j \in R_2} \text{cost}(sr_j)$

- …..a self-punctuating, horizon-based ranked join operator (binary, m-way)…
- …optimizations…
- Twig query processing over weighted data graphs

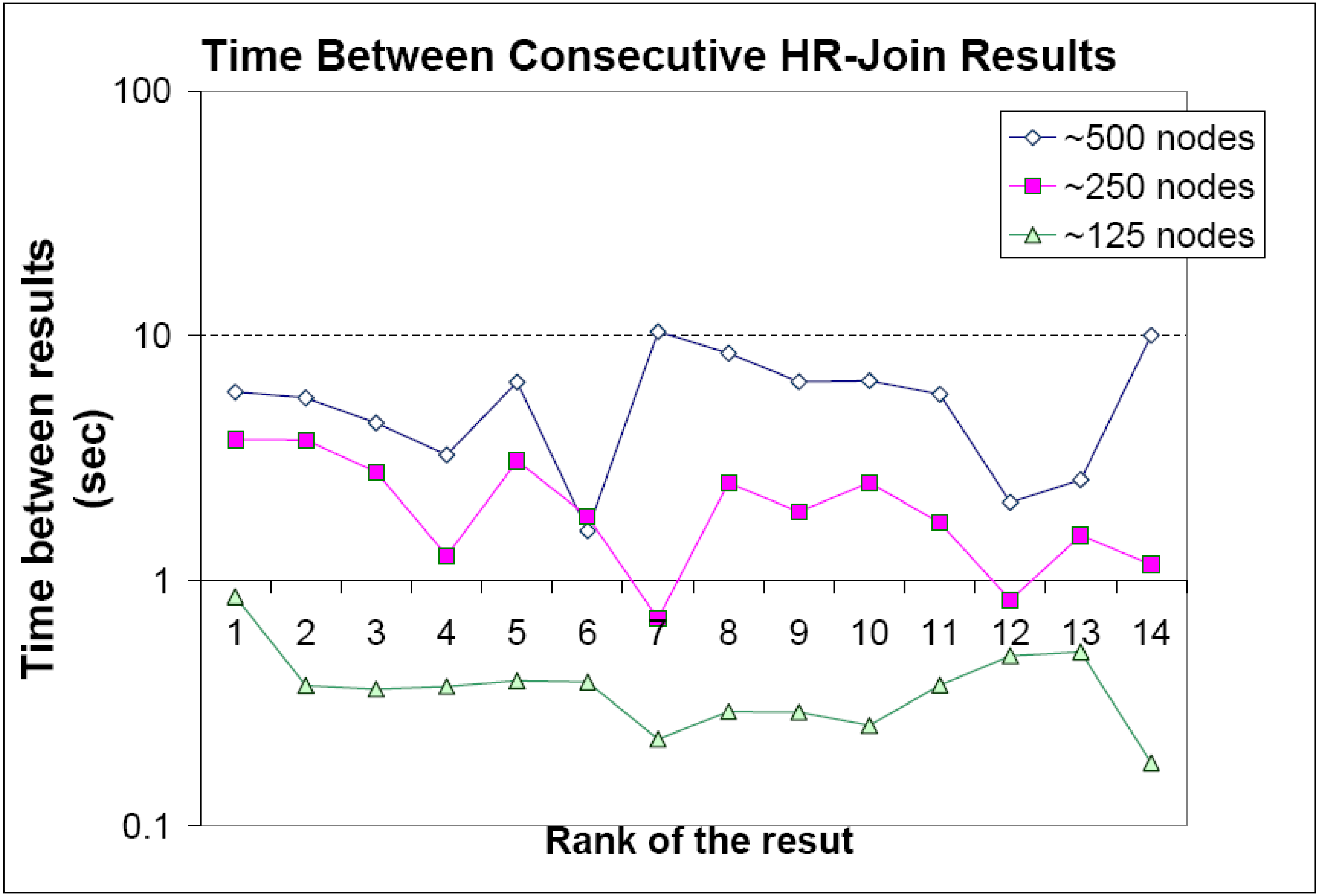
# Questions?



# The degree of pruning is directly correlated with the size of k



# The degree of input pruning is more important in bigger graphs


Time Between Consecutive HR-Join Results
Time between results (sec)
Rank of the resut
~500 nodes
~250 nodes
~125 nodes