Stream Processing
and
Incremental View Maintenance

Supplemental Reading
(papers posted on Piazza)
R&G Ch 3.6, Ch 25.8 - 25.10

Streaming Data

- **Databases**: Repeatedly process different queries on the same data.

- **Streaming/IVM**: Repeatedly process different data with the same queries.

Streaming Data Challenges

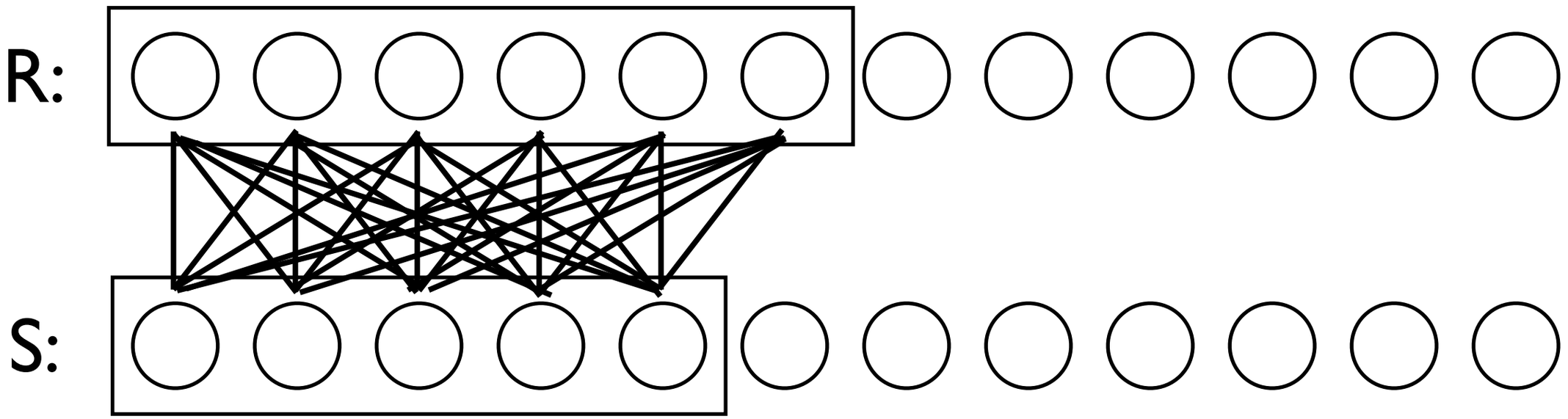
- **Blocking**: Streams are infinite, can't block until they finish.
  - Use Half Joins (Similar to Ripple Joins)
- **Scaling**: Insertion cost must be constant in the size (ripple joins are linear)
  - Use Window Joins
- **Performance**: Streaming data can arrive faster than the machine can handle.
  - Drop tuples as needed to maximize output rate.

WINDOW

```
SELECT L.state, T.month,
       AVG(S.sales) OVER W as movavg
FROM   Sales S, Times T, Locations L
WHERE  S.timeid = T.timeid
  AND  S.locid = L.locid
WINDOW W AS (
    PARTITION BY L.state
    ORDER BY T.month
    RANGE BETWEEN INTERVAL ‘1’ MONTH PRECEDING
          AND INTERVAL ‘1’ MONTH FOLLOWING
)
```

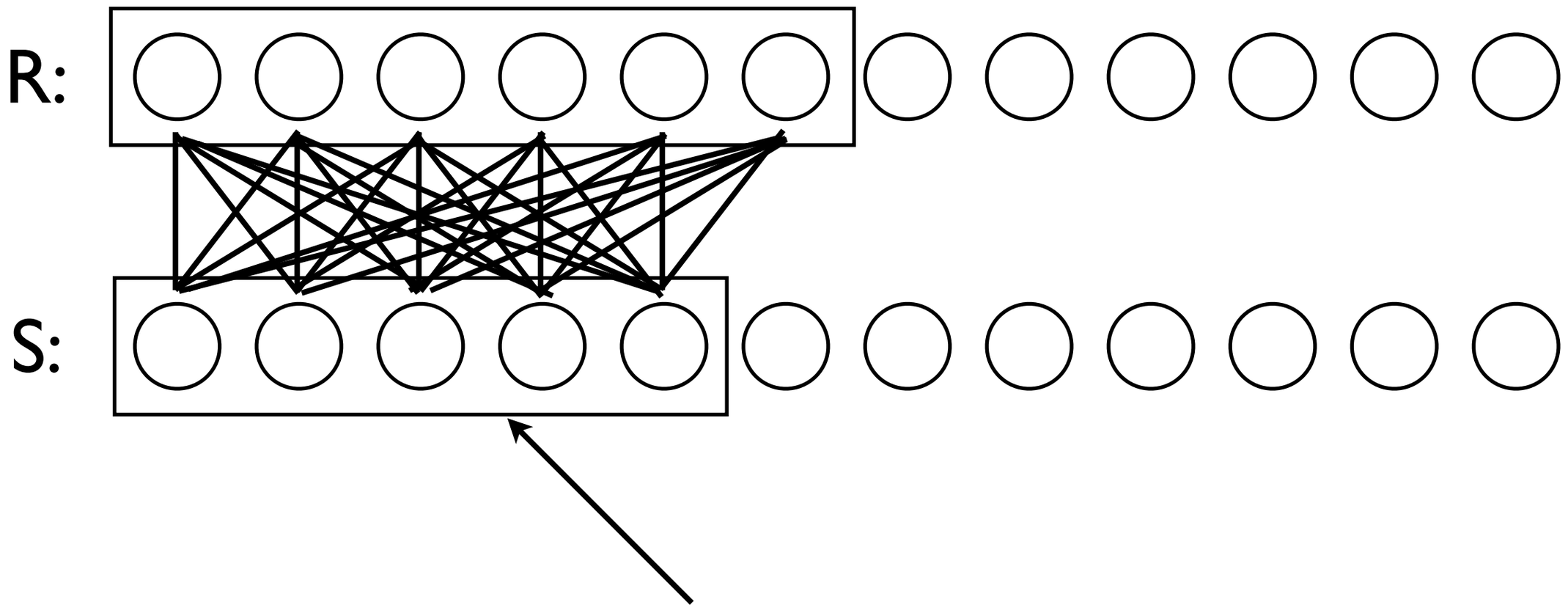
Window Joins

R ⋈ S



Window Joins

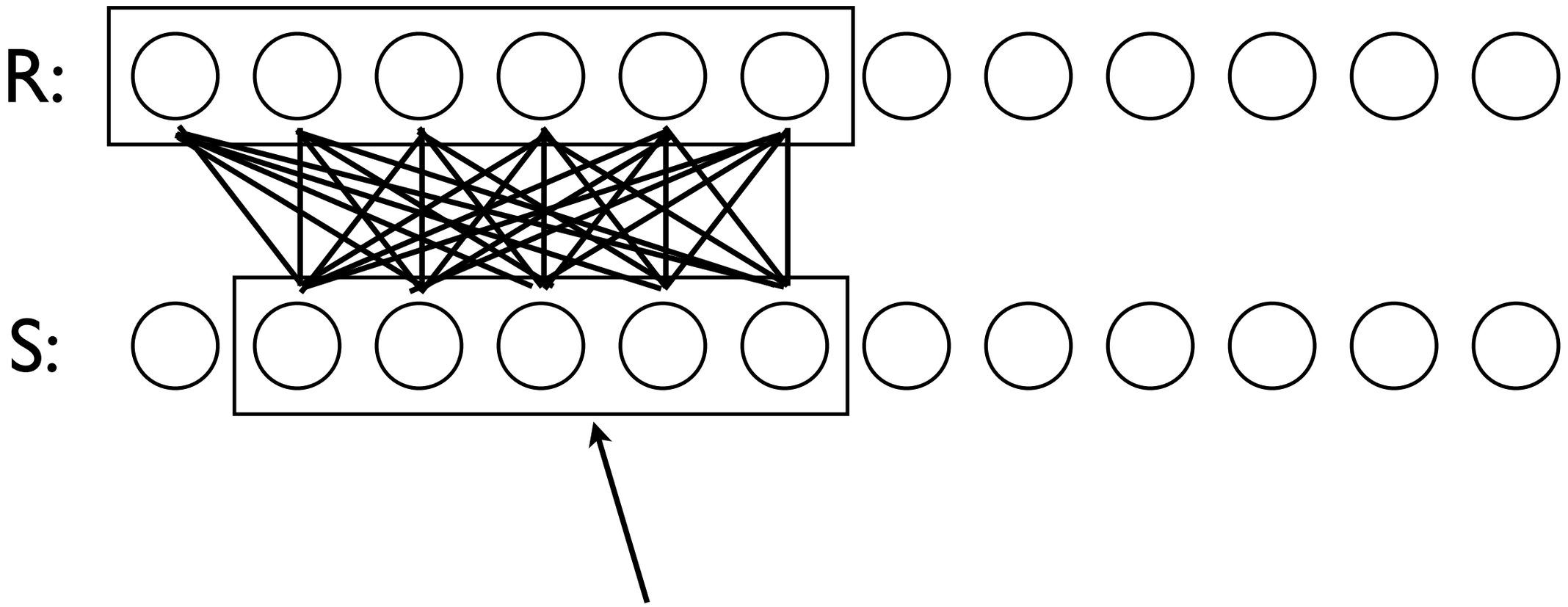
R ⋈ S



Do the full join for all tuples in the windows

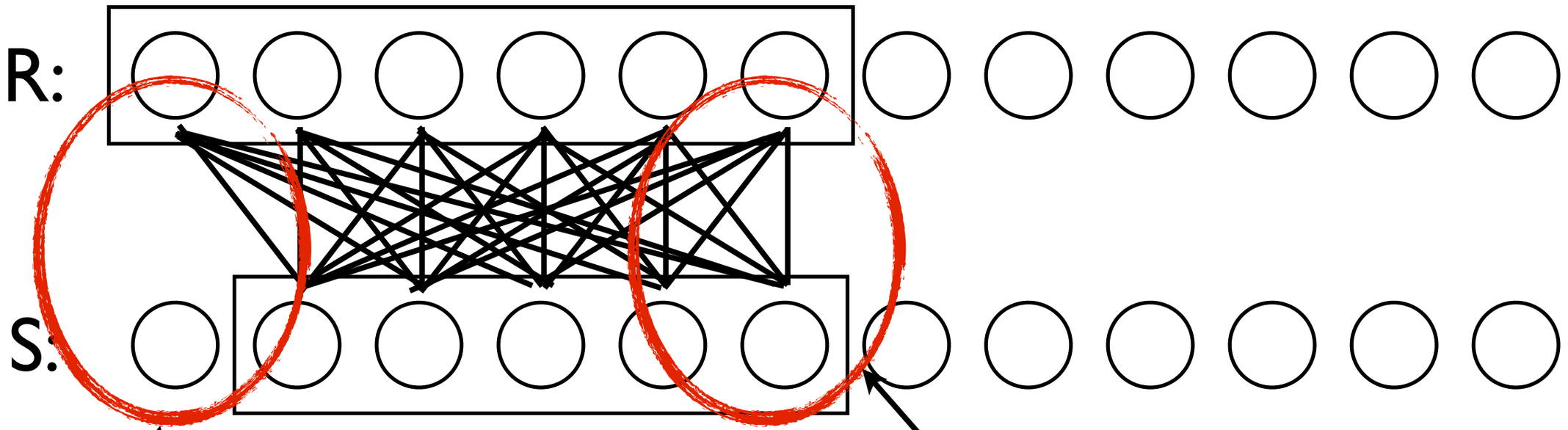
Window Joins

R ⋈ S



Window Joins

R ⋈ S

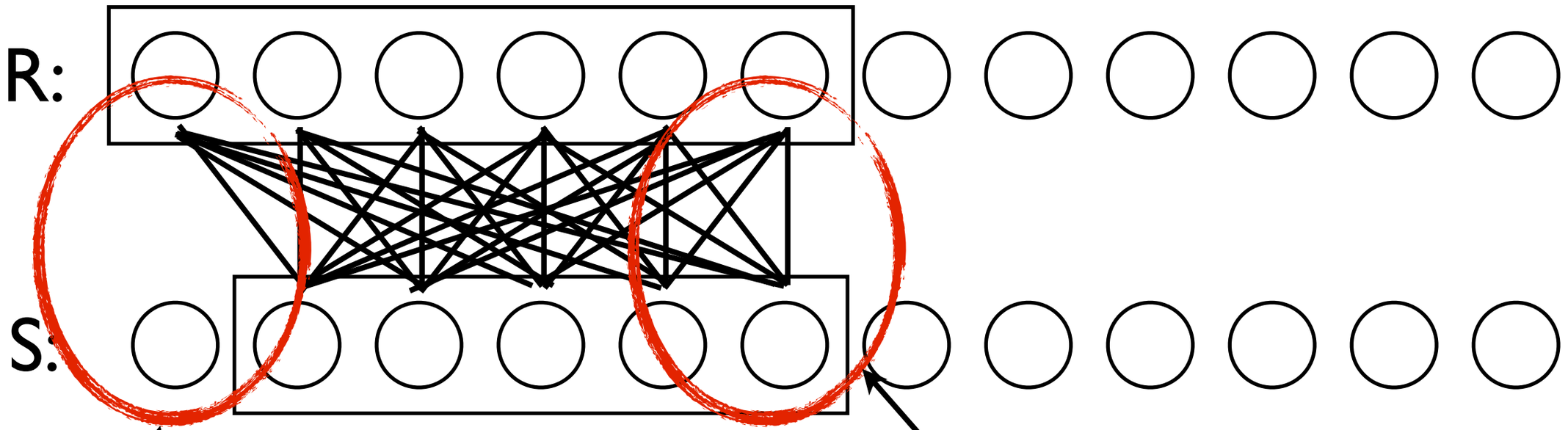


New Results: Tuples of R That Join With New S Tuple

Deleted Results: Tuples of R That Join With Expiring S Tuple

Window Joins

R ⋈ S



New Results: Tuples of R That Join With New S Tuple

~~**Deleted Results**: Tuples of R That Join With Expiring S Tuple~~

Streaming Window Joins

- Stream joins produce streams of new join outputs.
- **Problem**: We must support insertions into either stream, in any order.
- **Solution**: Use a Ripple Join
- **Problem**: Joins are still slow
- **Solution**: Maintain an index over the contents of each window.

Streaming Window Joins

- When a new tuple **t** arrives from R:
  - Use Index on S's Window to Join vs **t**.
  - Insert New Tuple into R's Window.
  - Delete Tuples Expiring From R's Window.

Streaming Window Joins

- Use Index on S's Window to Join vs **t**.
  - Hash Index (O(~1) lookup, == only)
  - B+Tree Index (O(log(n)) lookup, == or >)
  - No Index (O(n) lookup, any predicate)

Streaming Window Joins

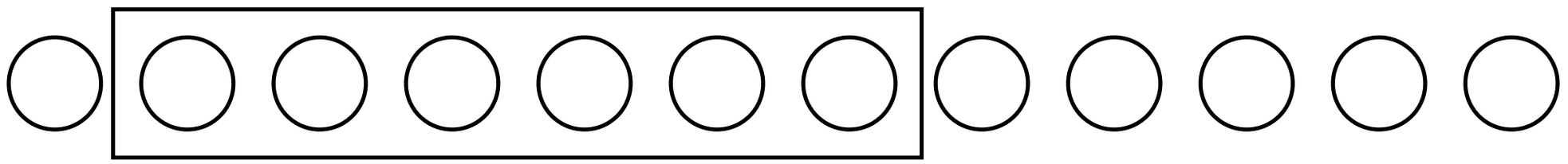
- Insert New Tuple into R's Window.
  - Easy. Standard Index Insertion.
- Delete Tuples Expiring From R's Window.
  - Hard(er). Need to Find Expiring Tuples.

Streaming Window Joins

- Insert New Tuple into R's ~~Window~~ Index.
  - Easy. Standard Index Insertion.
- Delete Tuples Expiring From R's Window.
  - Hard(er). Need to Find Expiring Tuples.

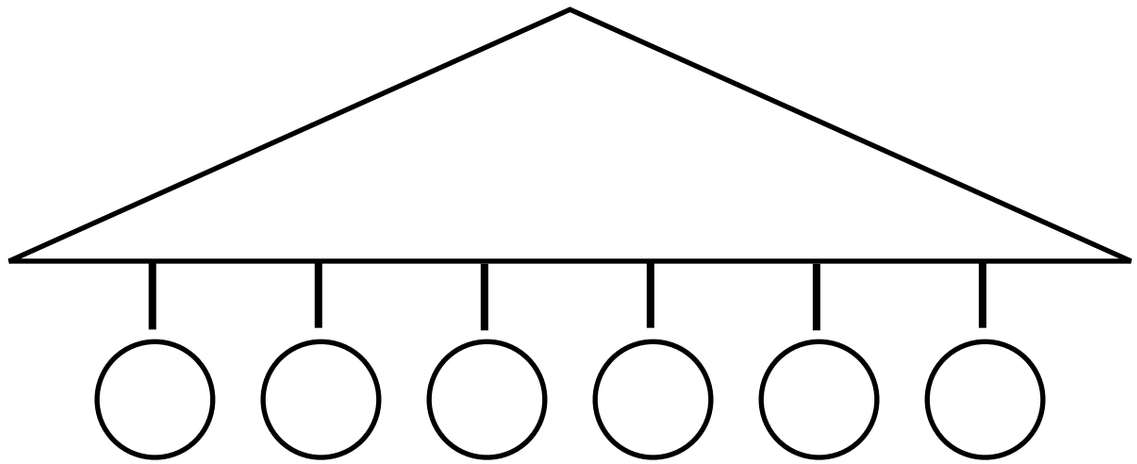
Invalidation-NLJ

Easy: Preserve Order of Tuples in Window



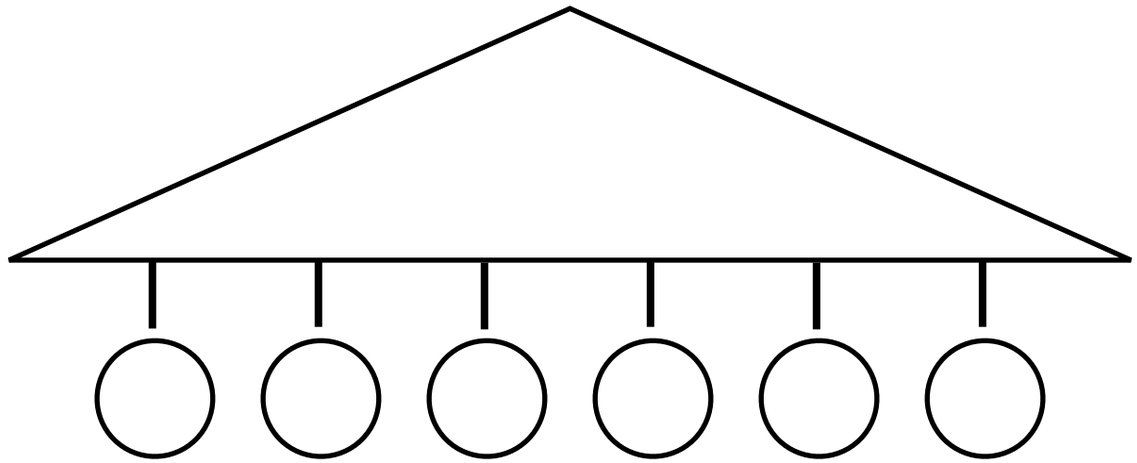
Only ever delete the last tuple(s)

Invalidation-Tree



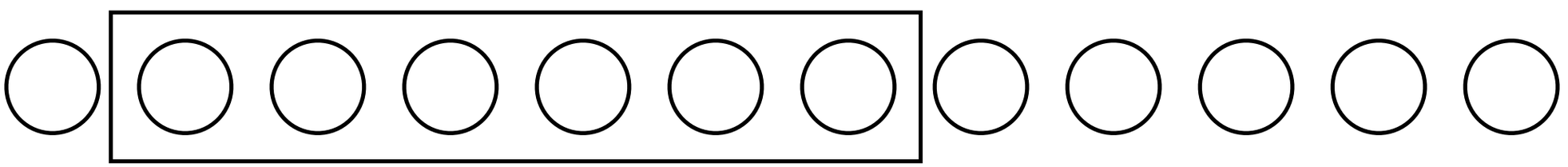
Problem: Tuples sorted in tree order

Invalidation-Tree

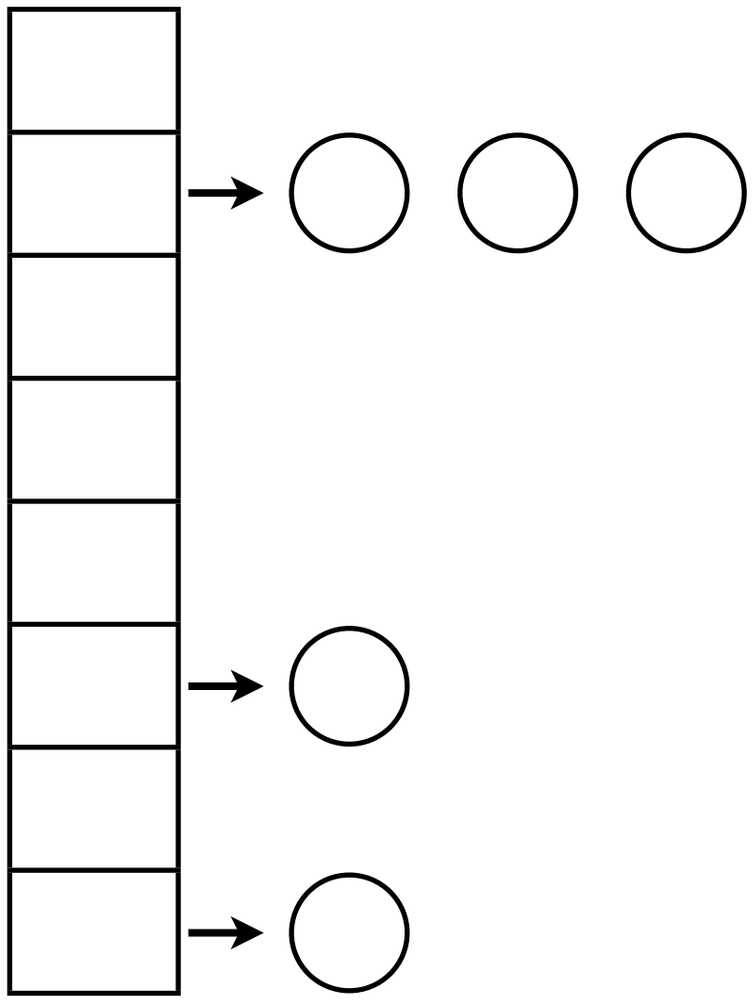


Problem: Tuples sorted in tree order

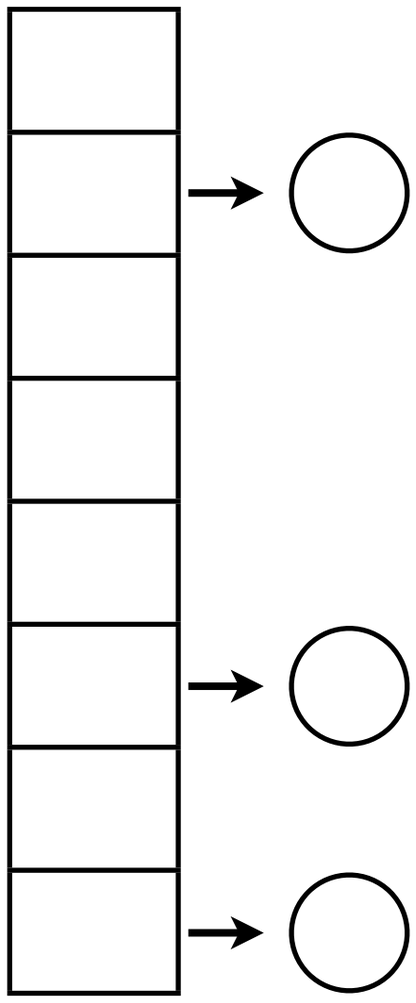
Solution: Maintain an array (like NLJ) in insertion order

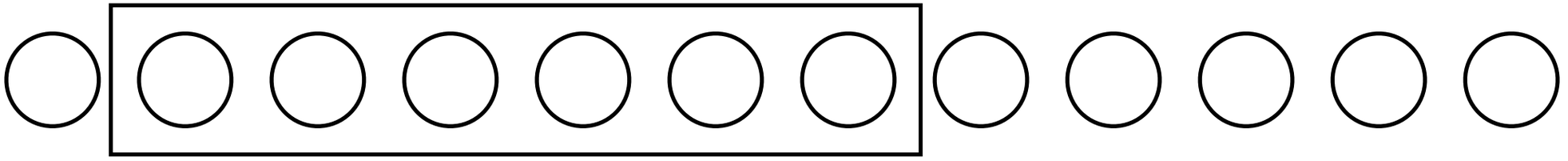


Invalidation-Hash



Invalidation-Hash

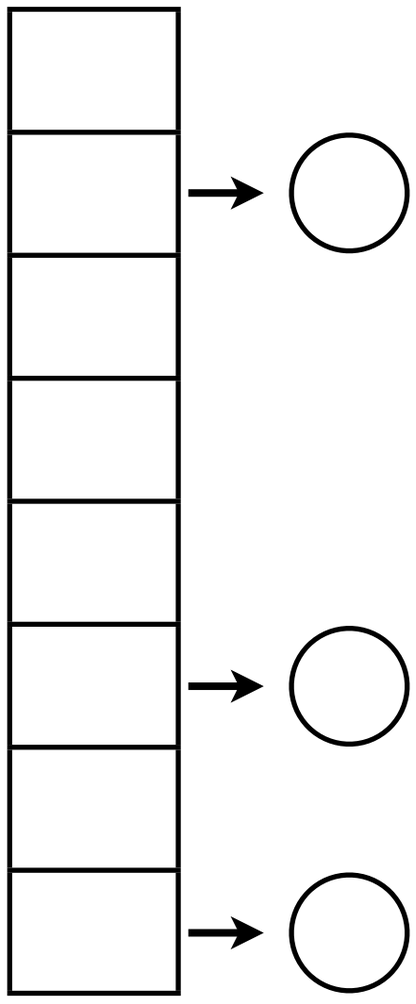




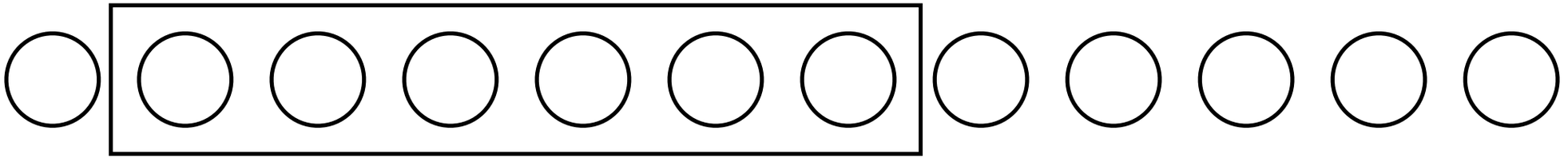
I: Keep Sorted List of Tuples in Window
(used to find hash bucket to delete from)

Invalidation-Hash

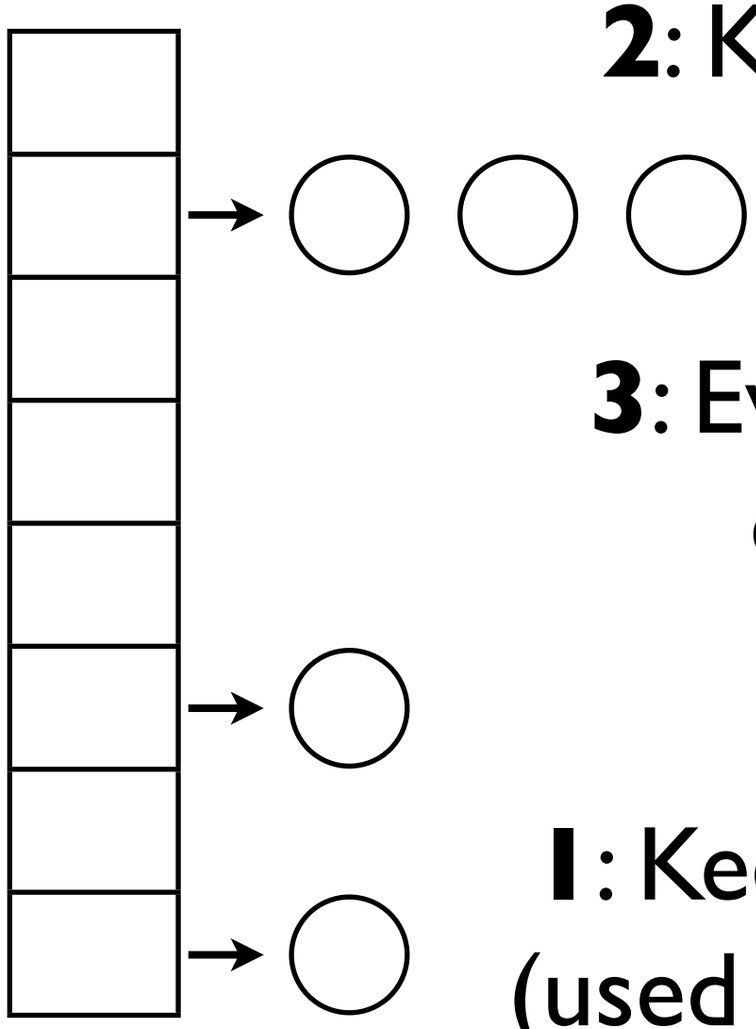
2: Keep Tuples In Each Bucket Sorted



1: Keep Sorted List of Tuples in Window
(used to find hash bucket to delete from)



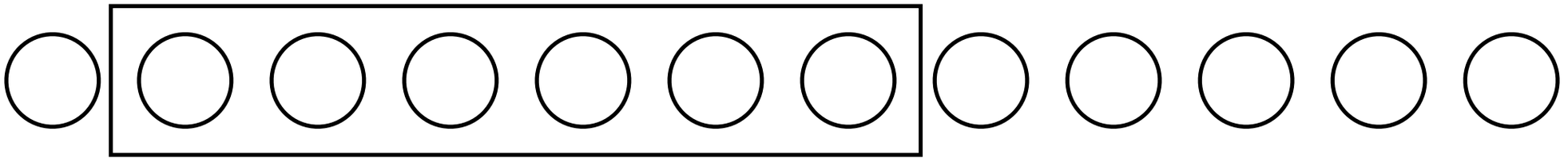
Invalidation-Hash



2: Keep Tuples In Each Bucket Sorted

3: Even If A Bucket Has >1 Item, O(1) deletion time for each bucket.

1: Keep Sorted List of Tuples in Window (used to find hash bucket to delete from)



Streaming Window Joins

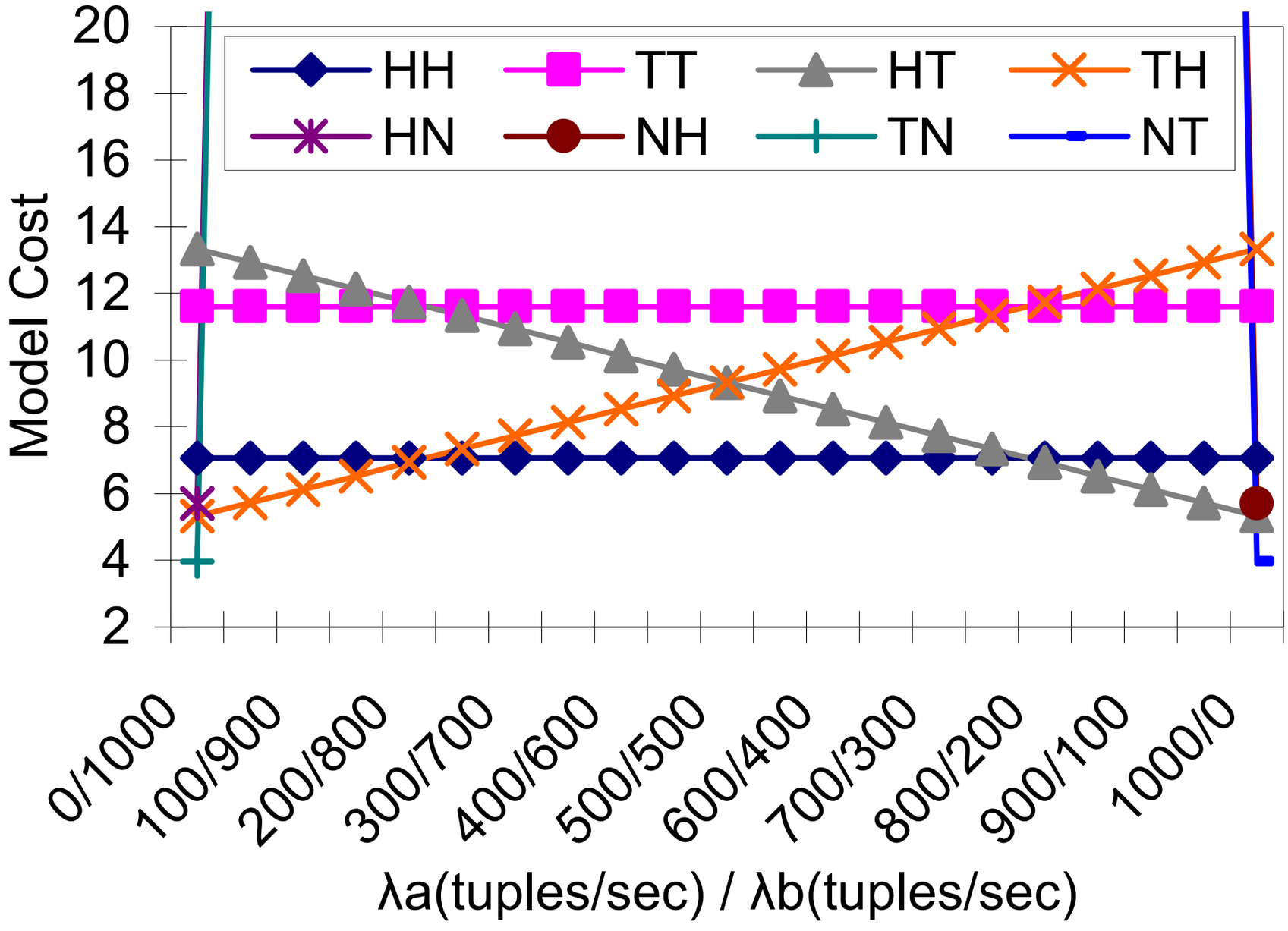
- The algorithms presented are sufficient to process half of a join (hence half join).
- The half join used for joins from R to S is independent of that for joins from S to R.
- Two half joins = A streaming join
- Choice of which half-join to use depends on *relative* insertion rates of R and S.

Streaming Window Joins

- **Recap**: When a new tuple **t** arrives from R:
  - Use Index on S's Window to Join vs **t**.
  - Insert New Tuple into R's Window.
  - Delete Tuples Expiring From R's Window.
- When would you ever use an NLJ?

Streaming Window Joins

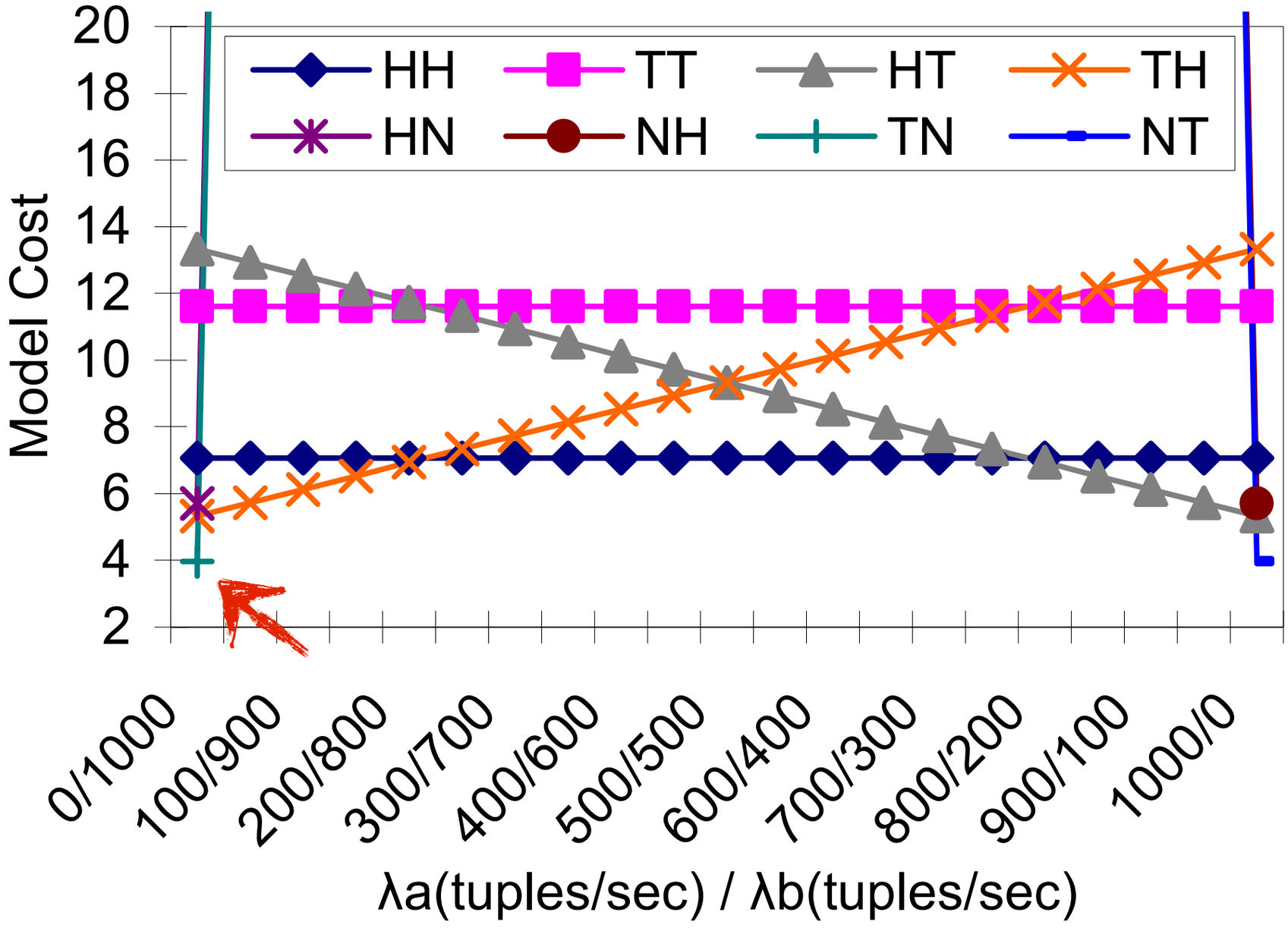
H = Hash
N = NLJ
T = T-Tree
(Similar To B+)



Cost Model In Paper

Streaming Window Joins

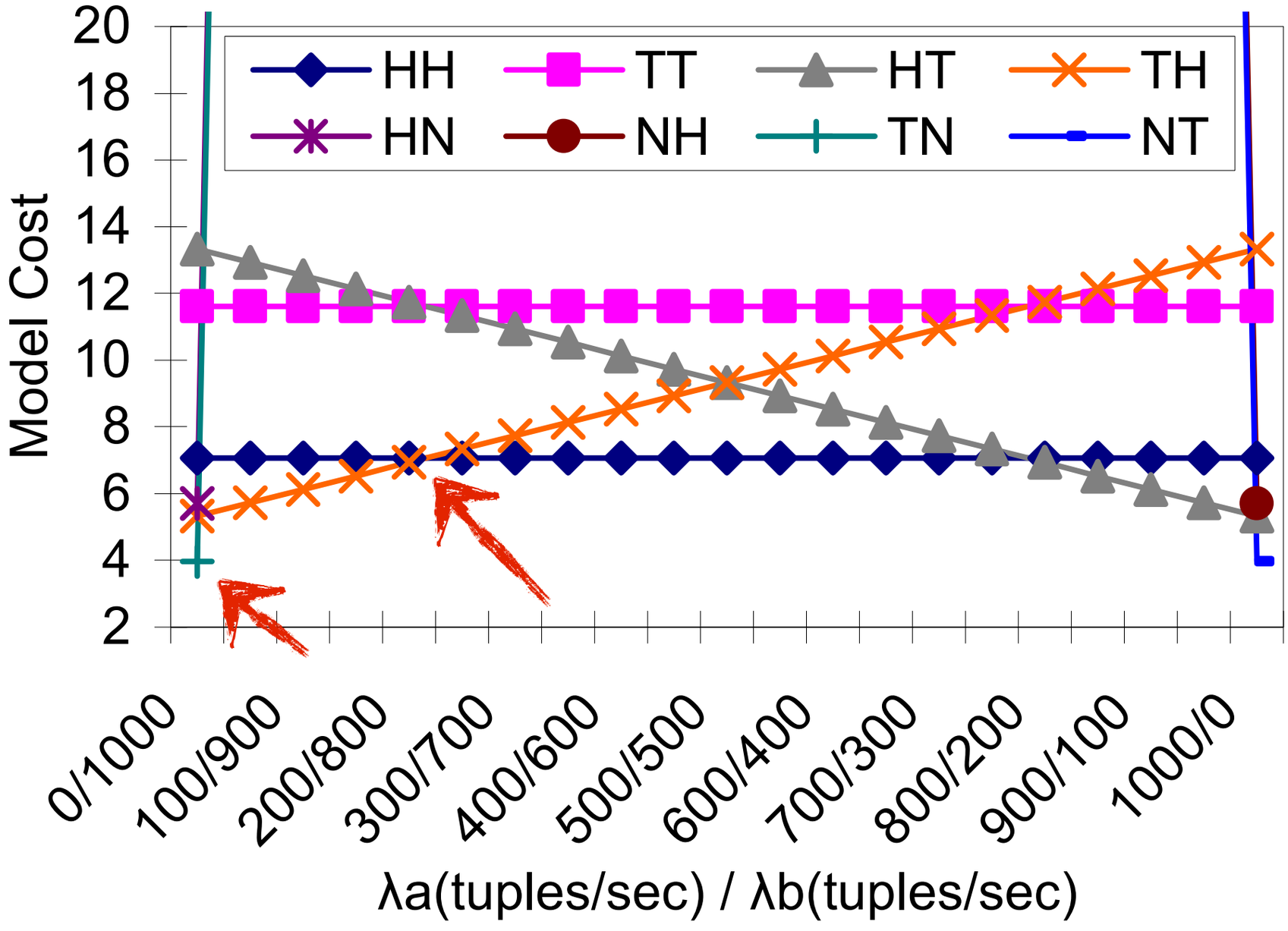
H = Hash
N = NLJ
T = T-Tree
(Similar To B+)



Cost Model In Paper

Streaming Window Joins

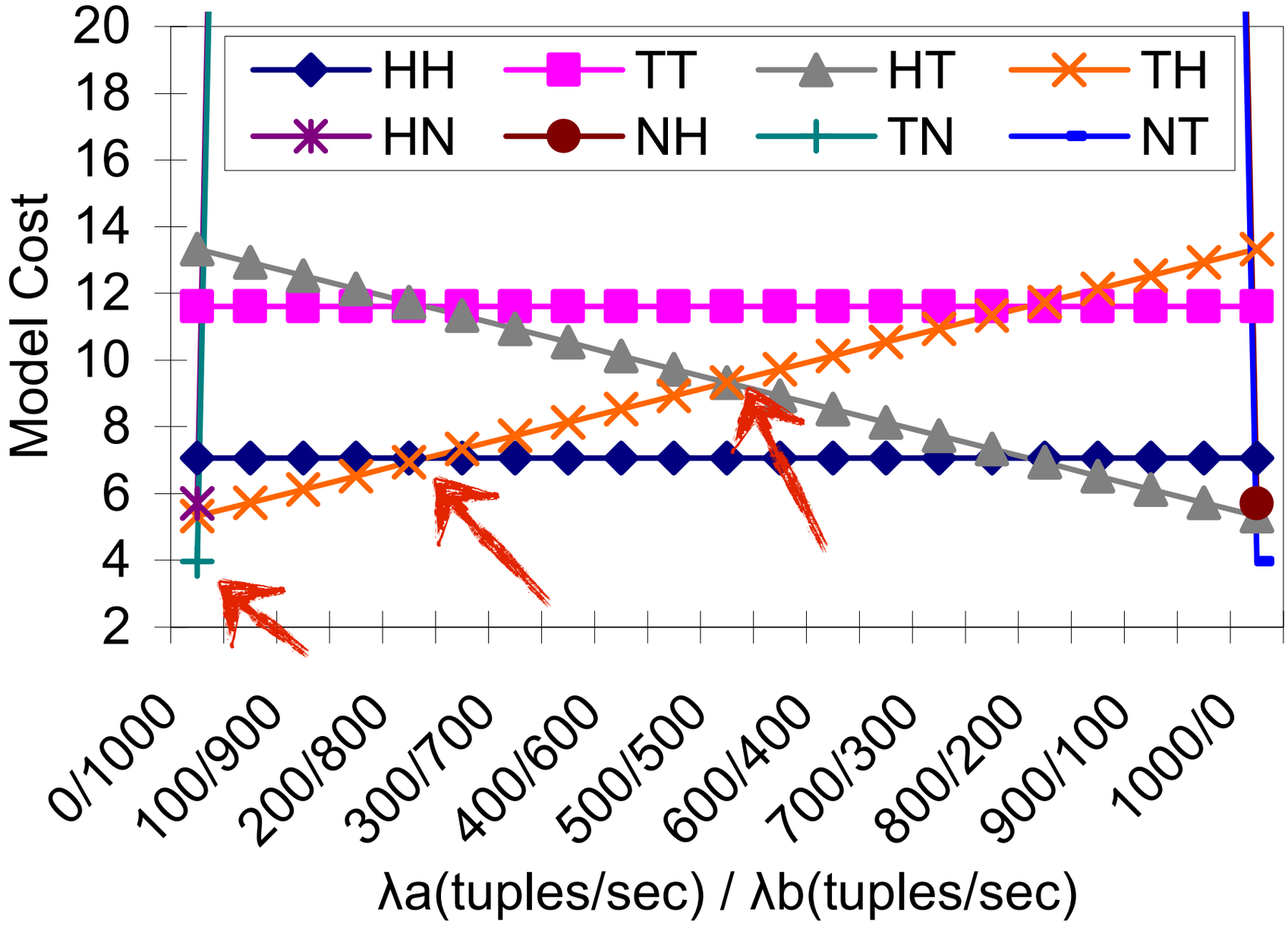
H = Hash
N = NLJ
T = T-Tree
(Similar To B+)



Cost Model
In Paper

Streaming Window Joins

H = Hash
N = NLJ
T = T-Tree
(Similar To B+)



Cost Model
In Paper

Views

```
CREATE VIEW EnterpriseOfficer(oid,name)
    AS SELECT o.oid, o.name
         FROM Officer o, Ship s
        WHERE o.ship = s.sid;
```

Views

```
CREATE VIEW EnterpriseOfficer(oid,name)
    AS SELECT o.oid, o.name
         FROM Officer o, Ship s
        WHERE o.ship = s.sid;
```

```
SELECT eo.name
FROM EnterpriseOfficer eo;
```

Views

```
CREATE VIEW EnterpriseOfficer(oid,name)
    AS SELECT o.oid, o.name
         FROM Officer o, Ship s
        WHERE o.ship = s.sid;
```

```
SELECT eo.name
FROM (SELECT o.oid, o.name
             FROM Officer o, Ship s
            WHERE o.ship = s.sid
      ) eo;
```

Materialized Views

- **Problem**: View Queries Can be Expensive
- **Solution**: Save the View Query Result as a normal table.
- **Problem**: What if the Data Changes
  - Re-evaluate the full query?
  - **Solution**: Delta Queries

Delta Queries

- If D is your Database and Q is your Query:
  - Q(D) is the result of your query on the database.
- Let's say you make a change ΔD (Insert tuple)
  - Q(D+ΔD) is the new result
- If we have Q(D), can we get Q(D+ΔD) faster?
  - Analogy to Sum: {34, 29, 10, 15} + {12} (88+12)

Delta Queries

Delta Queries

σ

|

R

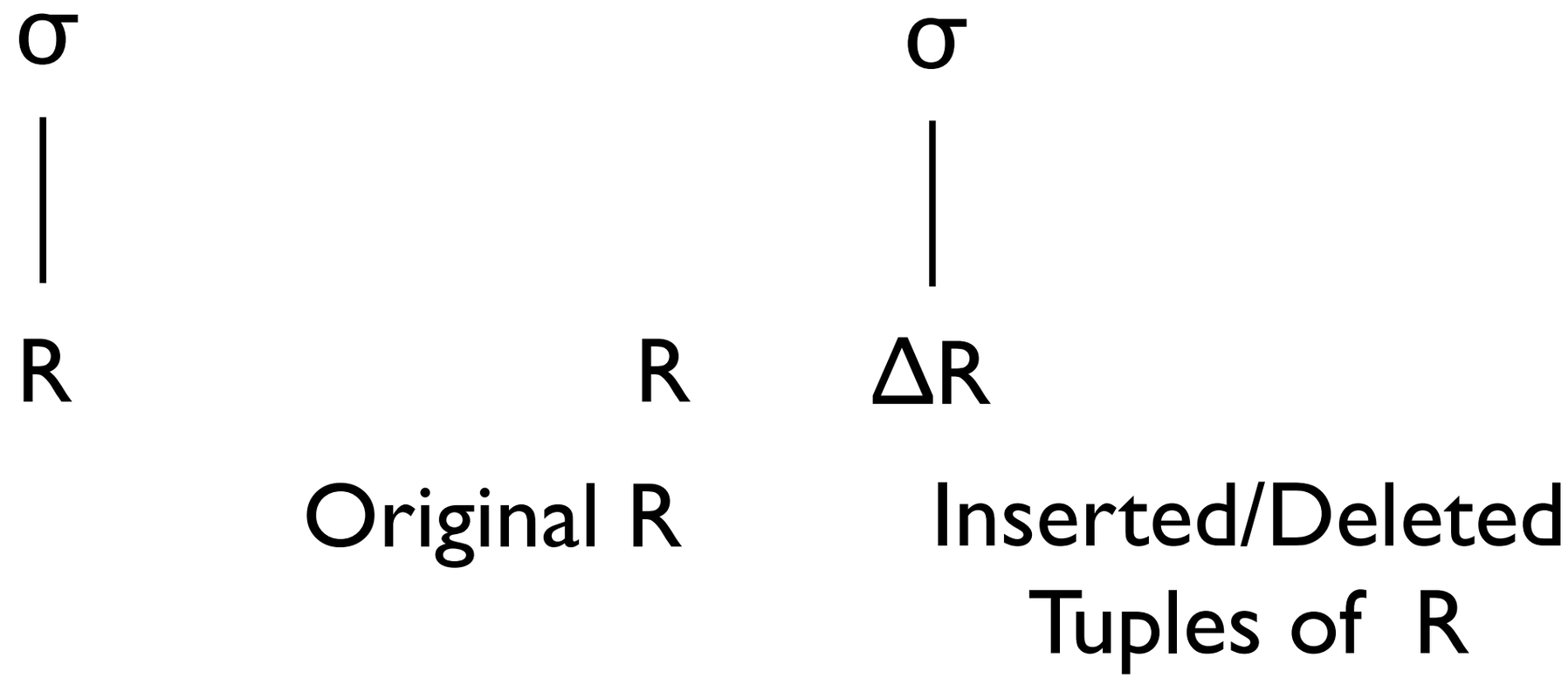
R

Original R

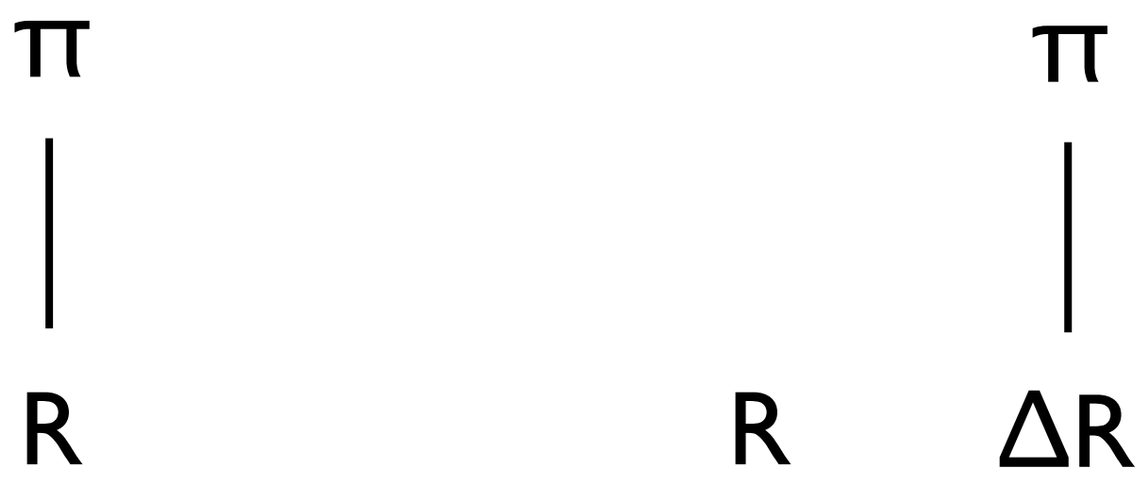
ΔR

Inserted/Deleted
Tuples of R

Delta Queries



Delta Queries



Delta Queries

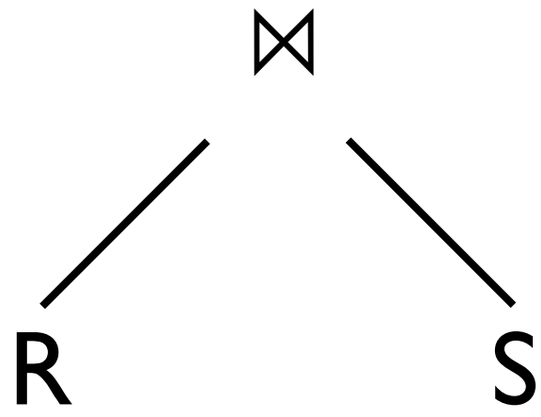
Problem: What about Set-Semantics Projection?

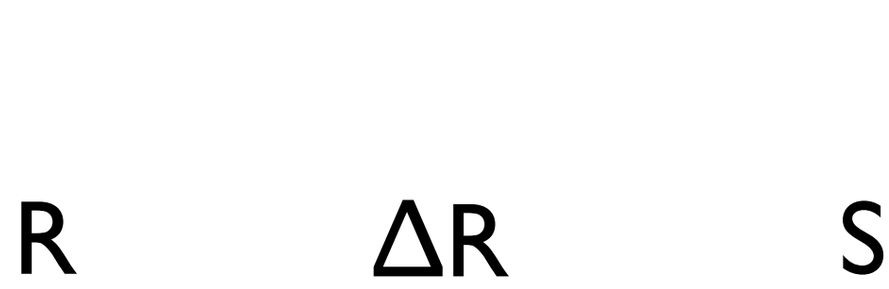
Delta Queries

Problem: What about Set-Semantics Projection?

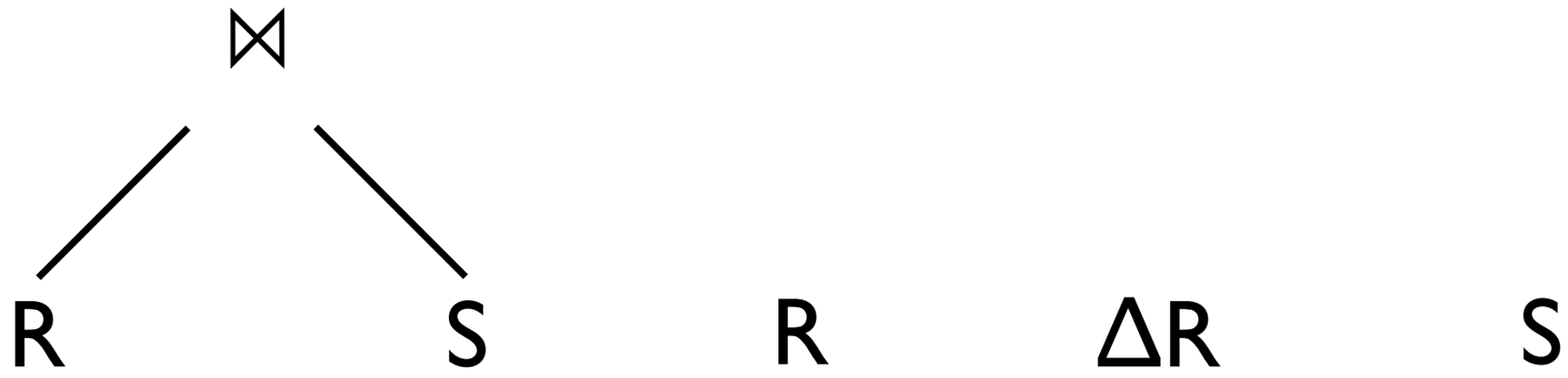
Problem: What about Deletion?

Delta Queries



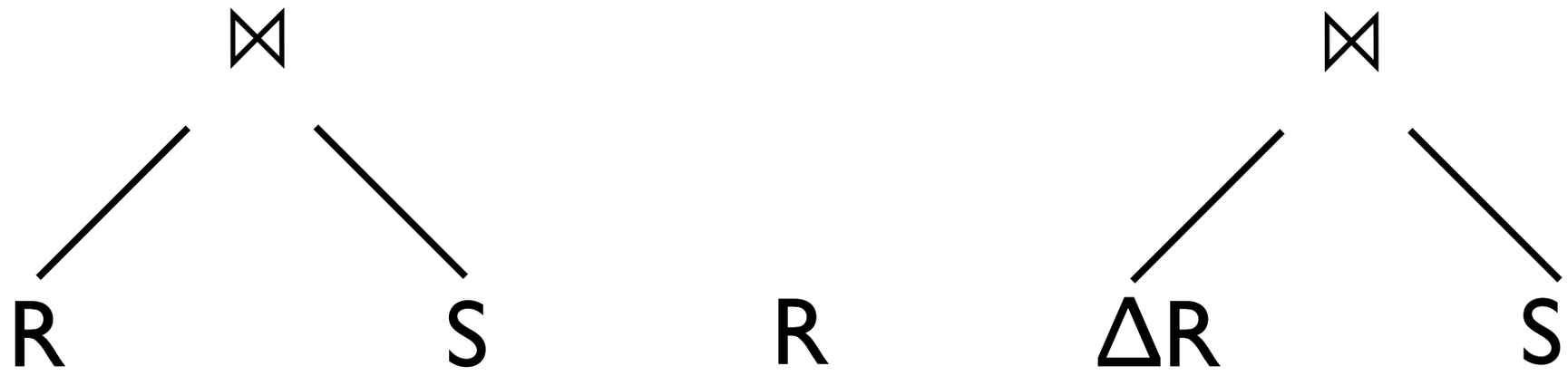


Delta Queries



Problem: What about Deletion?

Delta Queries



Problem: What about Deletion?

Delta Queries

What about aggregates?