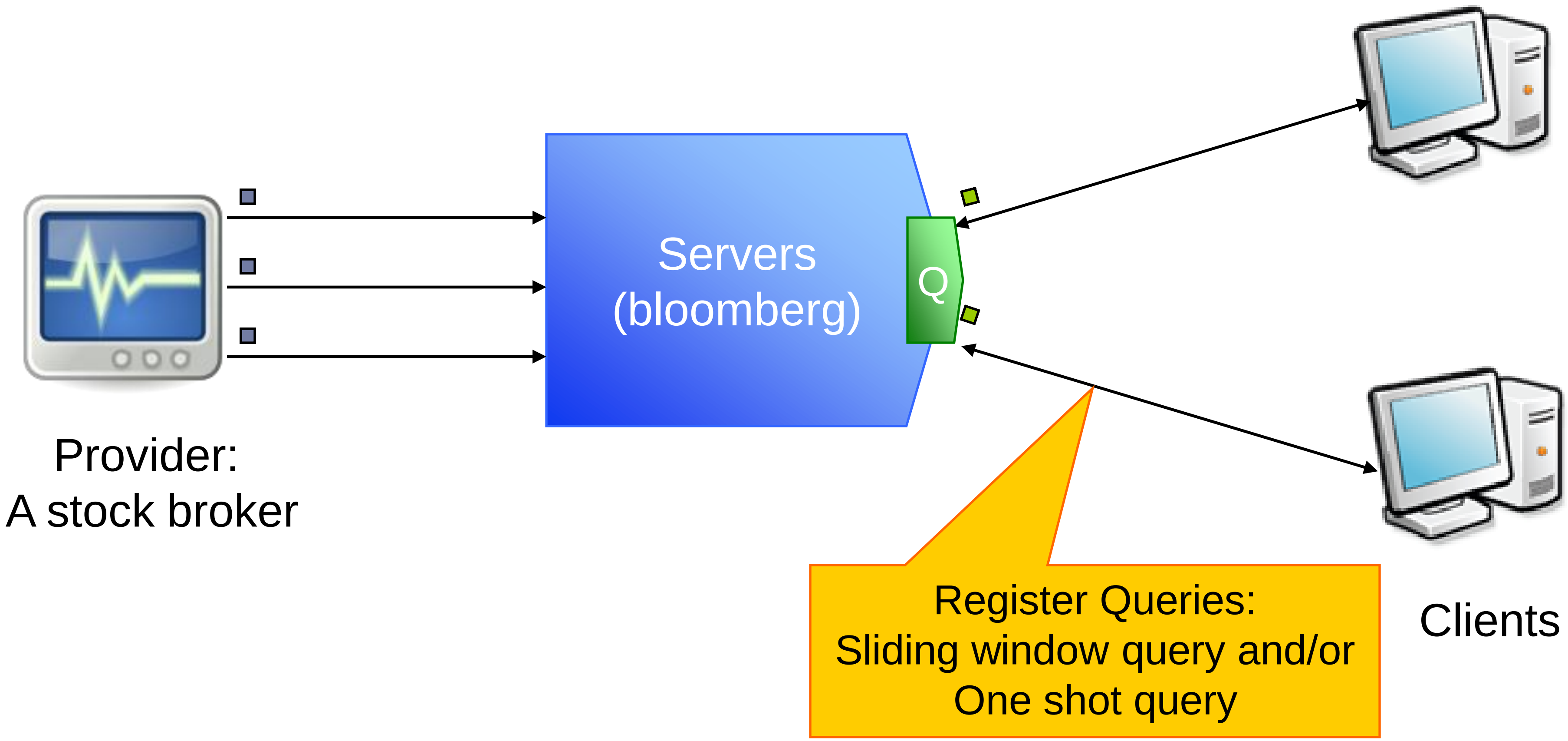
Proof-Infused Streams: Authenticating Sliding Window Queries on Data Streams

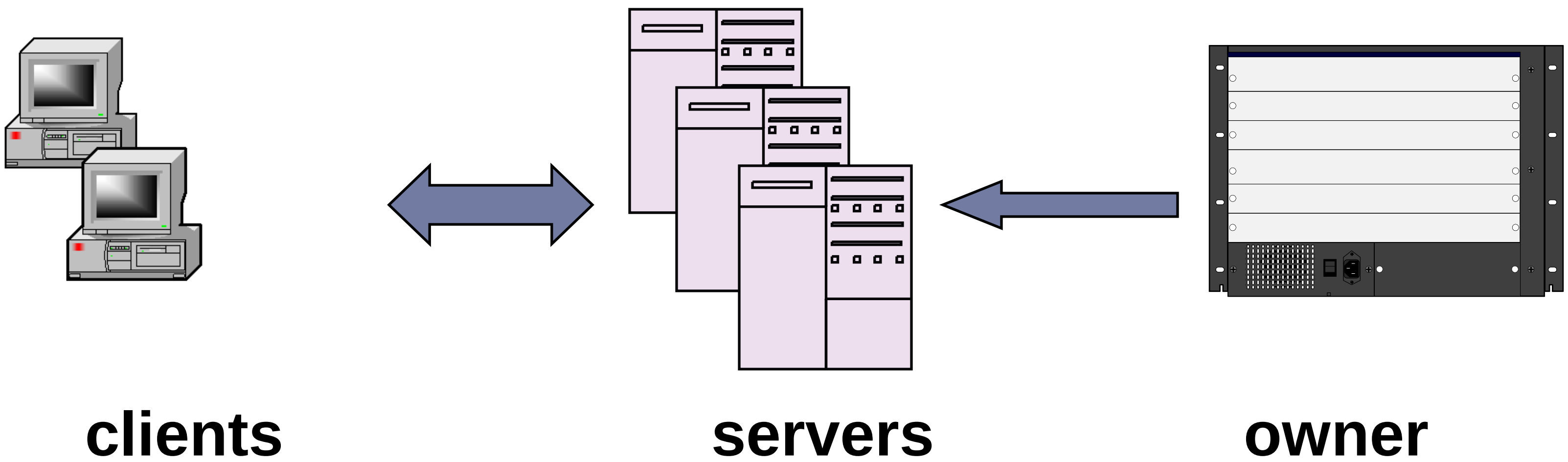
- Feifei Li, Florida State University
- Ke Yi, Hong Kong University of Science & Technology
- Marios Hadjieleftheriou, AT&T Labs Research
- George Kollios, Boston University

Outsourced stream model: stock trading monitoring



Data Publishing Model [HIM02]

Owner: publish data
Servers: host (or monitor) the data and provide query services
Clients: query the owner's data through servers



H. Hacigumus, B. R. Iyer, and S. Mehrotra, ICDE02

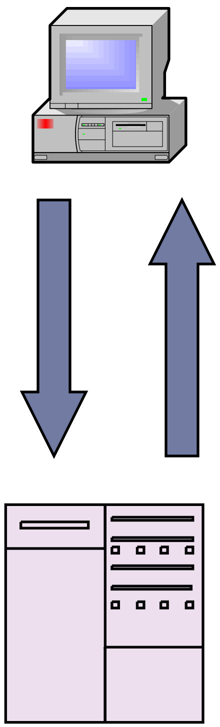
Information Security Issues

- The third-party (server) cannot be trusted
  - Lazy server
  - Malicious intent
  - Compromised equipment
  - Unintentional errors (e.g. bugs)

Problem 1: Injection

Select * from T where 5<A<11

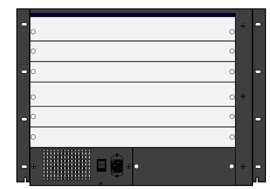
client



server

	A	B
r_1	…	
…	…	
r_{i-1}	4	
r_i	7	
r_{i+1}	9	
r_{i+2}	11	

owner

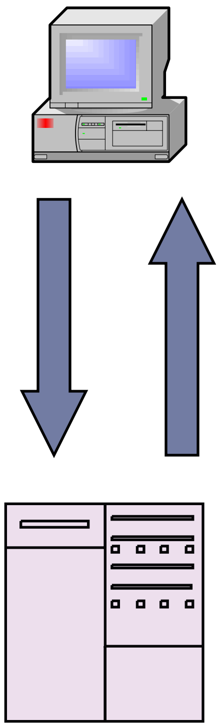


	A	B
r_1	…	
…	…	
r_{i-1}	4	
r_i	7	
r_{i+1}	9	
r_{i+2}	11	

Problem 2: Drop

Select * from T where 5<A<11

client

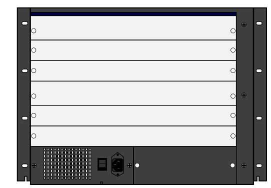


server

Returns 7

	A	B
r_1	…	
…	…	
r_{i-1}	4	
r_i	7	
r_{i+1}	9	
r_{i+2}	11	

owner



	A	B
r_1	…	
…	…	
r_{i-1}	4	
r_i	7	
r_{i+1}	9	
r_{i+2}	11	

Query Authentication: Goals

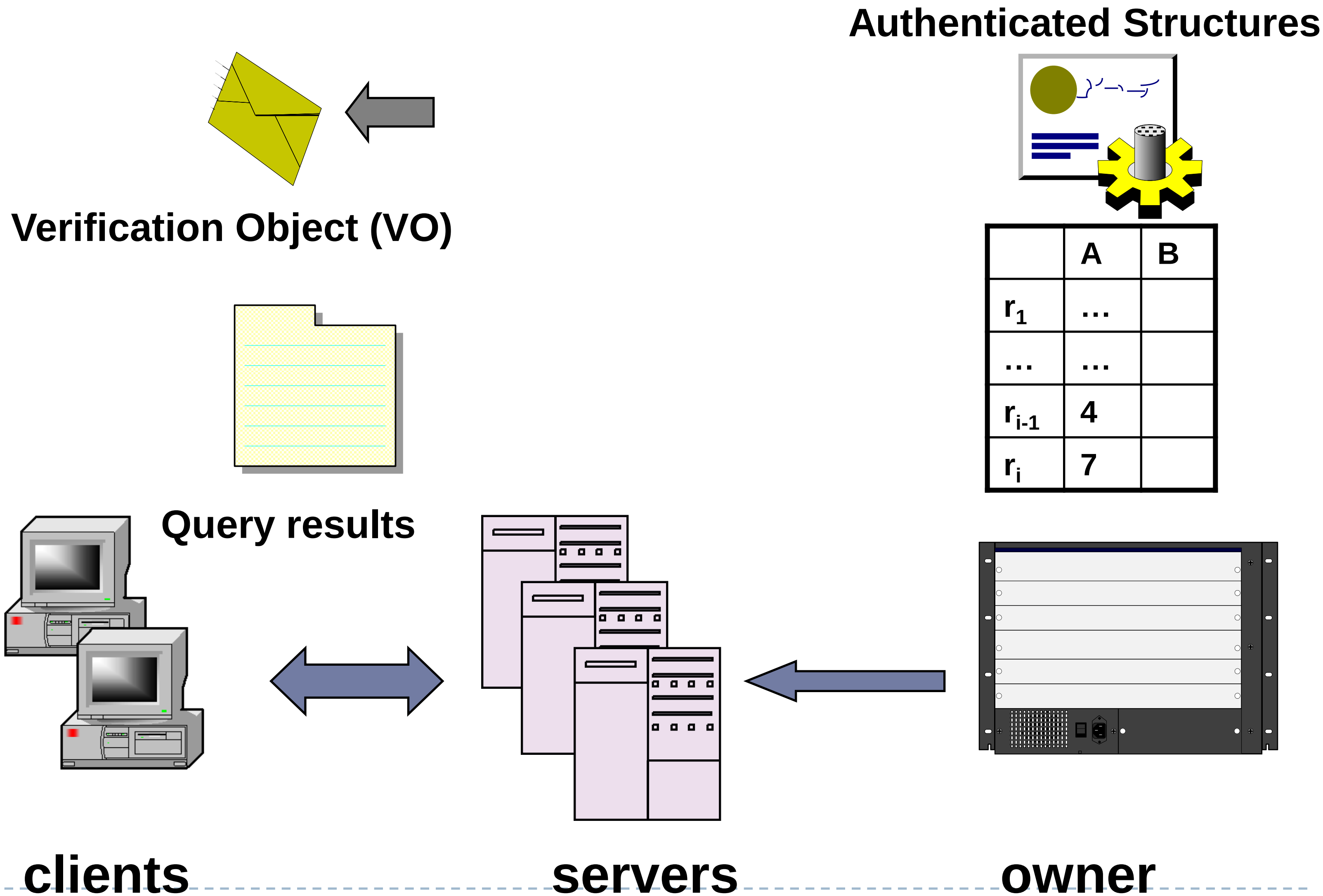
- Query Correctness

  results do exist in the owner's database

- Query Completeness

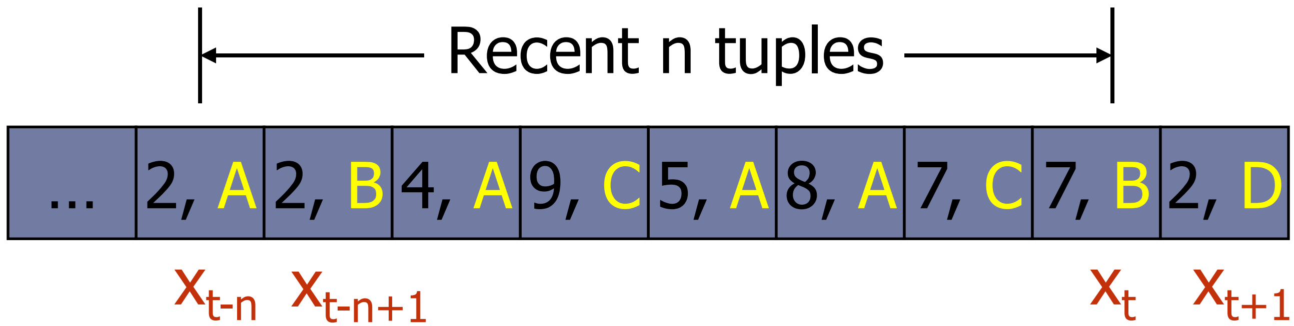
  no records have been omitted from the result

General Approach



	A	B
r_1	…	
…	…	
r_{i-1}	4	
r_i	7	

Sliding Window Query



Tuple-based Window

Time-based Window

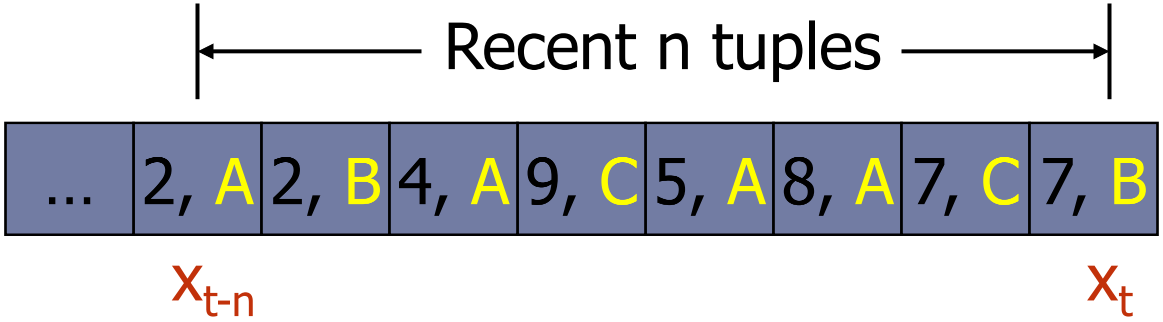
This talk concentrates on tuple-baesd window, generalizing to time-based window is in the paper.

For tuple-based window, the timestamp is simply the arrival id of the tuple.

```
SELECT SUM(stock_price)
FROM stock_trace
WHERE stock_name = A in last 50 Trades
SLIDES every 1 trade
```

```
SELECT SUM(stock_price)
FROM stock_trace
WHERE stock_name = A in last 5 Minutes
SLIDES every 1 minute
```

One Shot Query



Tuple-based Window

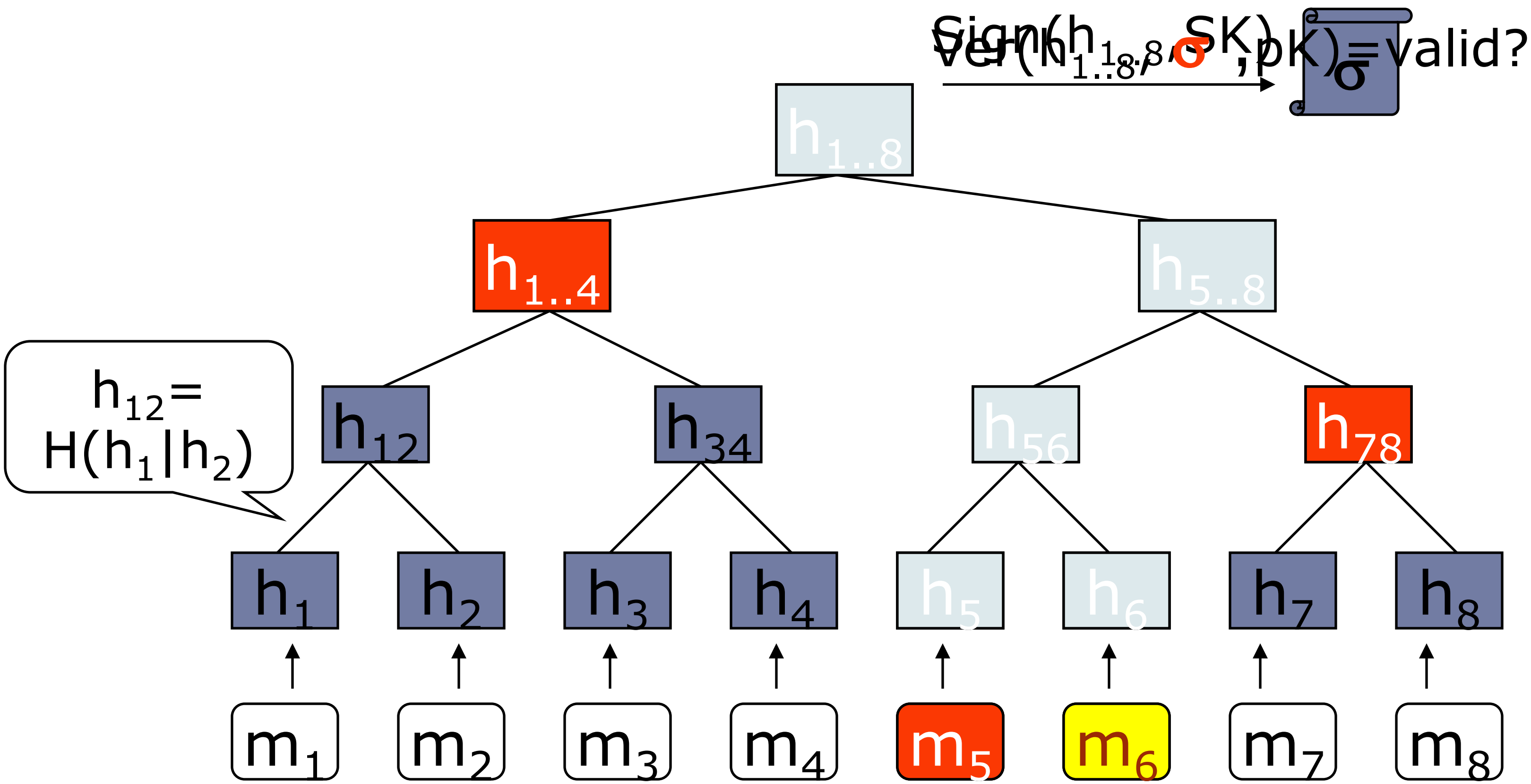
```
SELECT SUM(stock_price)
FROM Stock_trace
WHERE stock_name = A in last 100 Trades
```

Merkle Hash Tree[M89]-Amortizing Signature Cost

Single signature to sign many messages

Hash function is publicly known

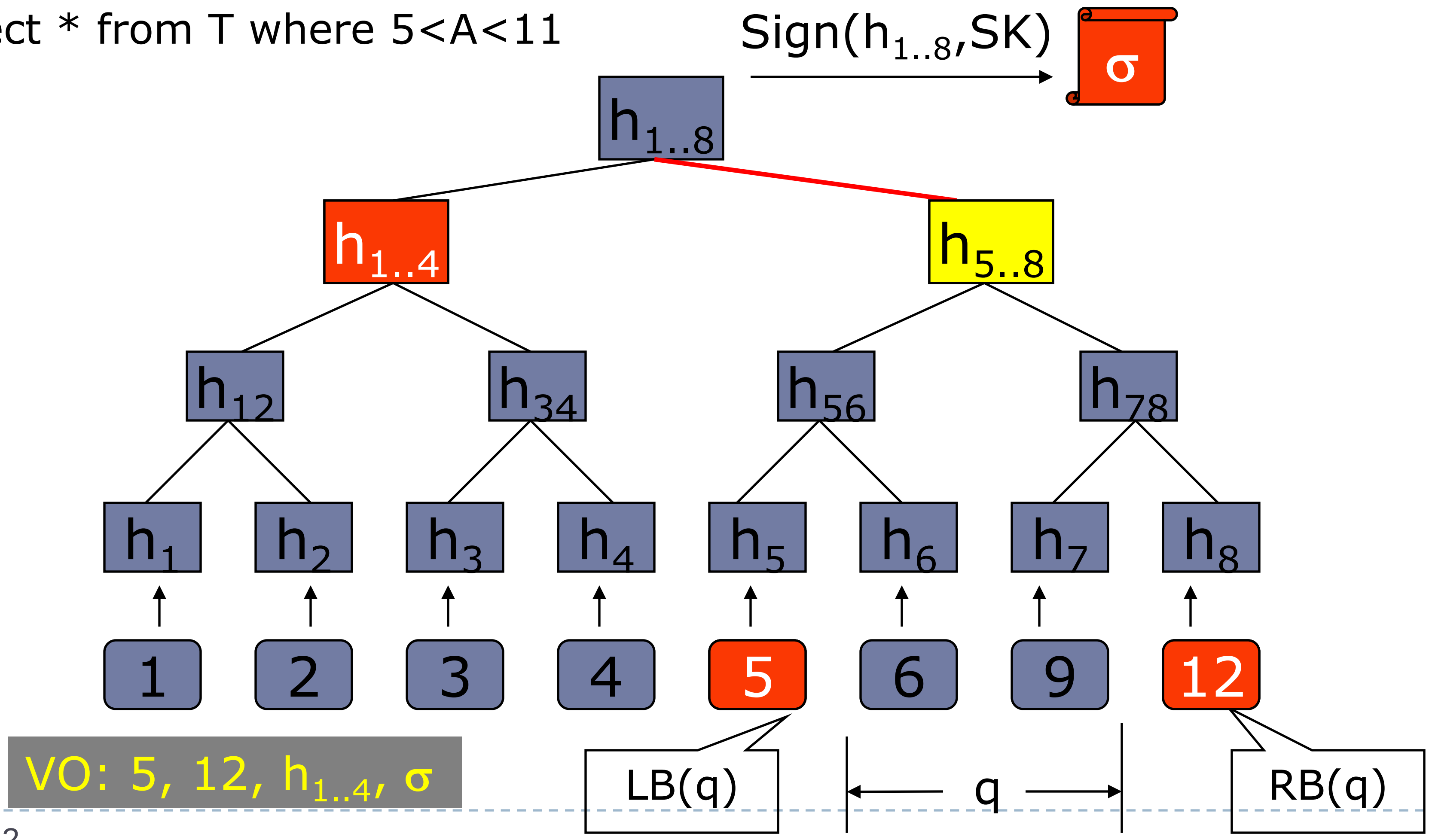
Collision resistant hash function → no one except the owner could produce a different hash value for the root



R. C. Merkle. CRYPTO, 1989

Extends to Range Query: f=2 (f is the fanout)

Select * from T where 5<A<11

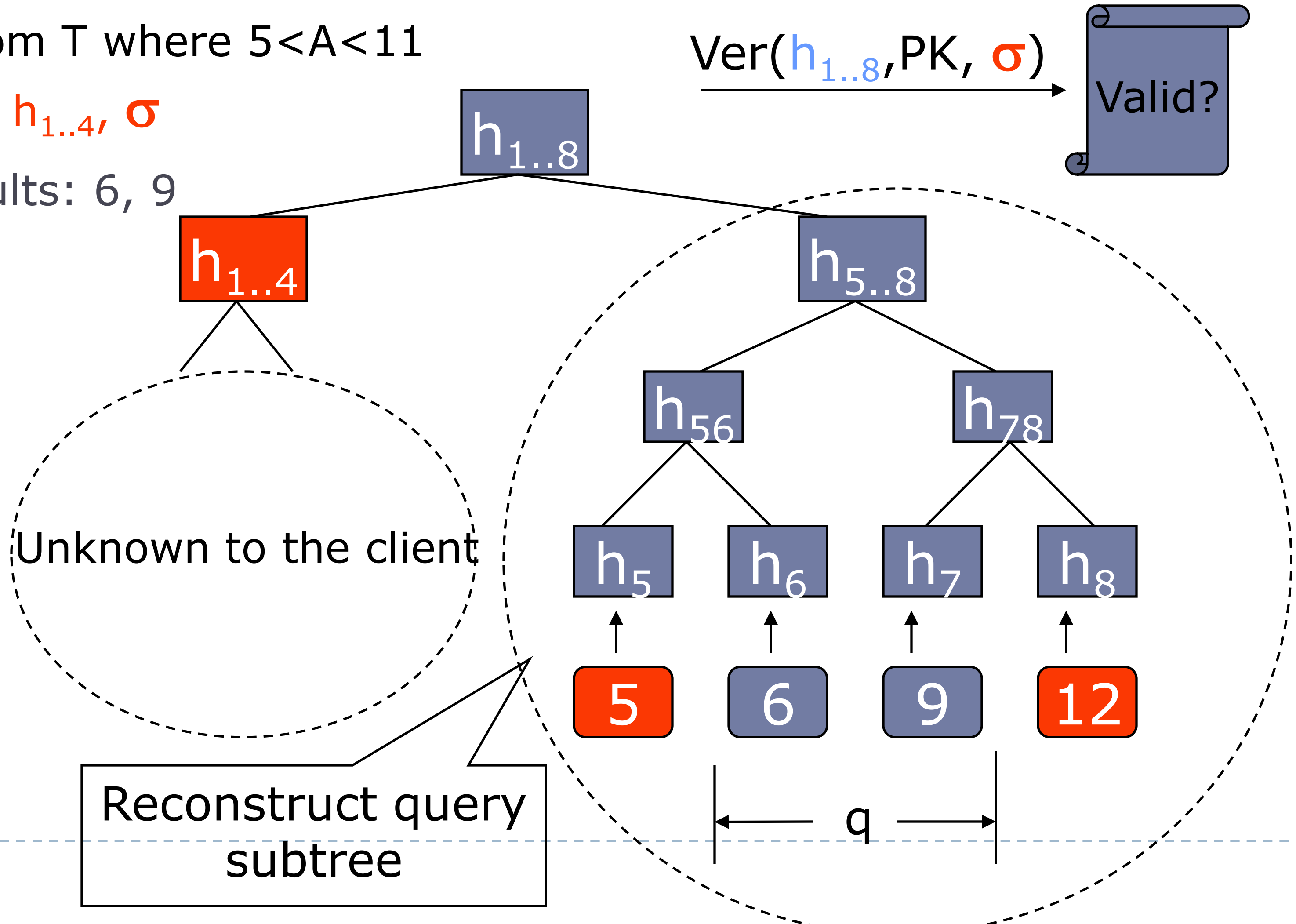


Client Side Verification

Select * from T where 5<A<11

VO: 5, 12, $h_{1..4}$, σ

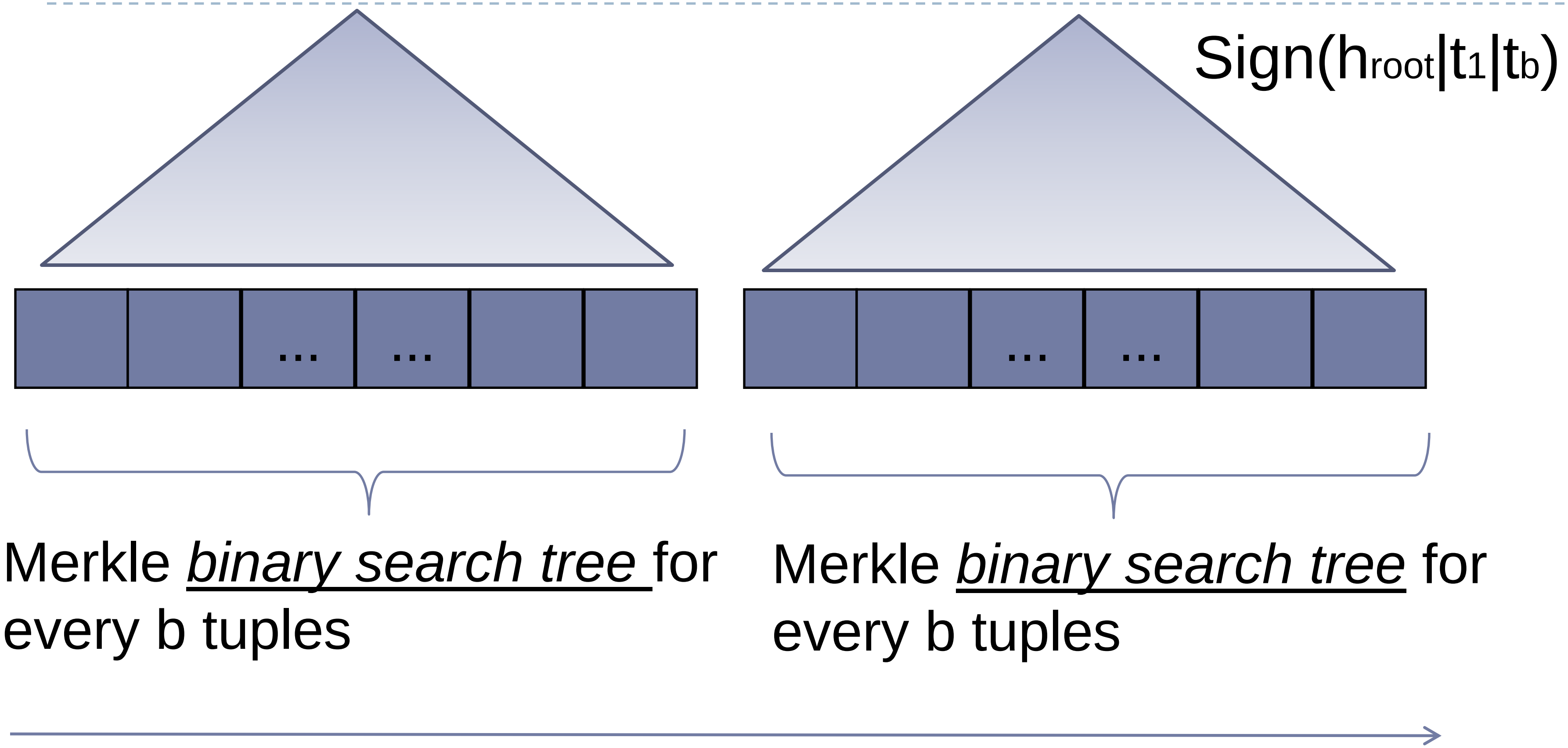
Query results: 6, 9



Solution Overview

- Sign Every Tuple (with query attribute(s) and timestamp)
  - Expensive update cost for the data provider
  - Expensive communication cost between server and clients as VO size is large
  - But it provides timely answer on a per-tuple basis
- Amortize the signing cost by "proof-infusing" on a group of tuples:
  - A delayed response, can often be tolerated.
- Query with d query attributes is a query in d+1 dimension.
- N: maximum window size; n: window size for a particular query; b: the delay

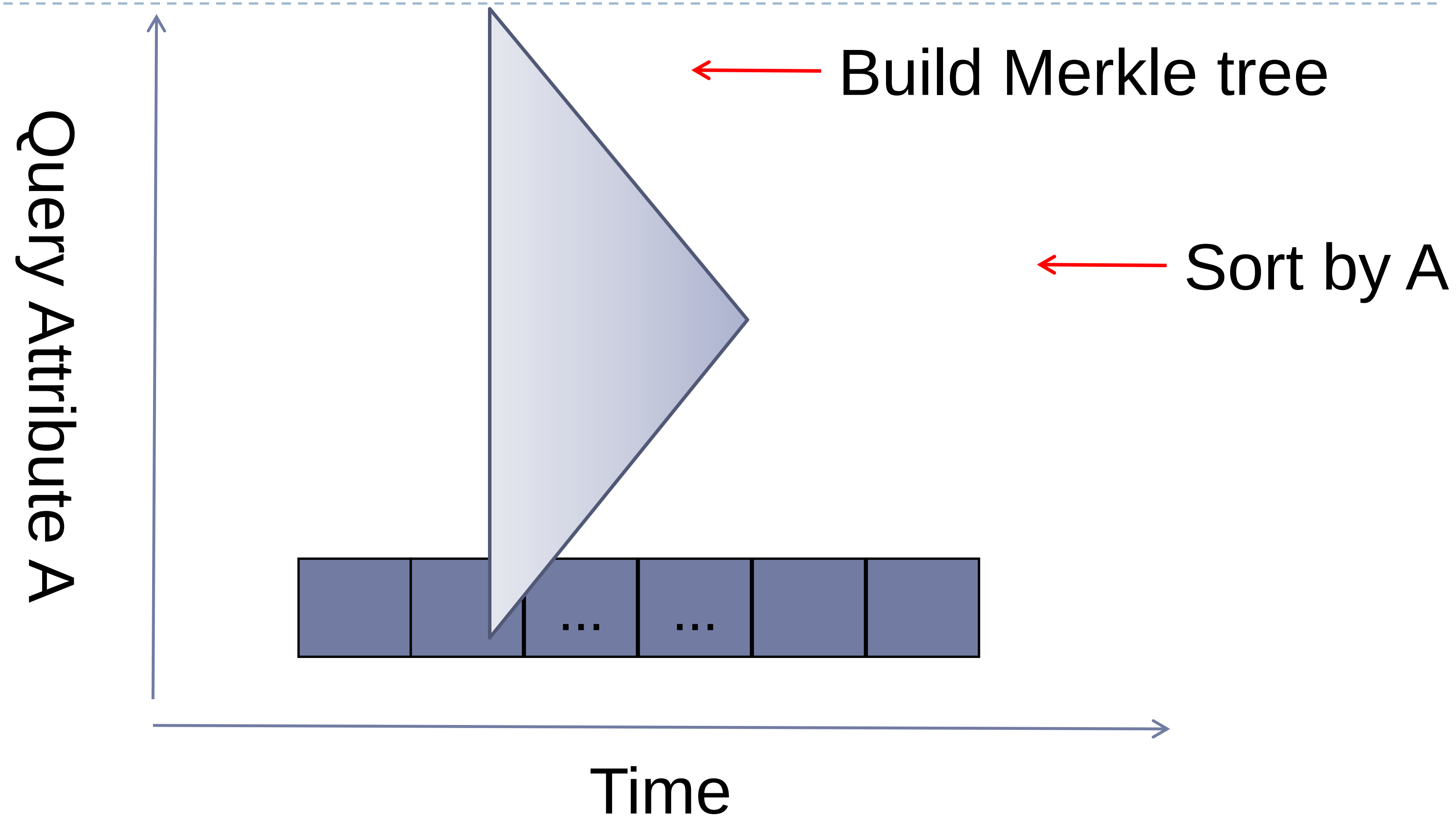
Tumbling Merkle Tree (TM-tree)



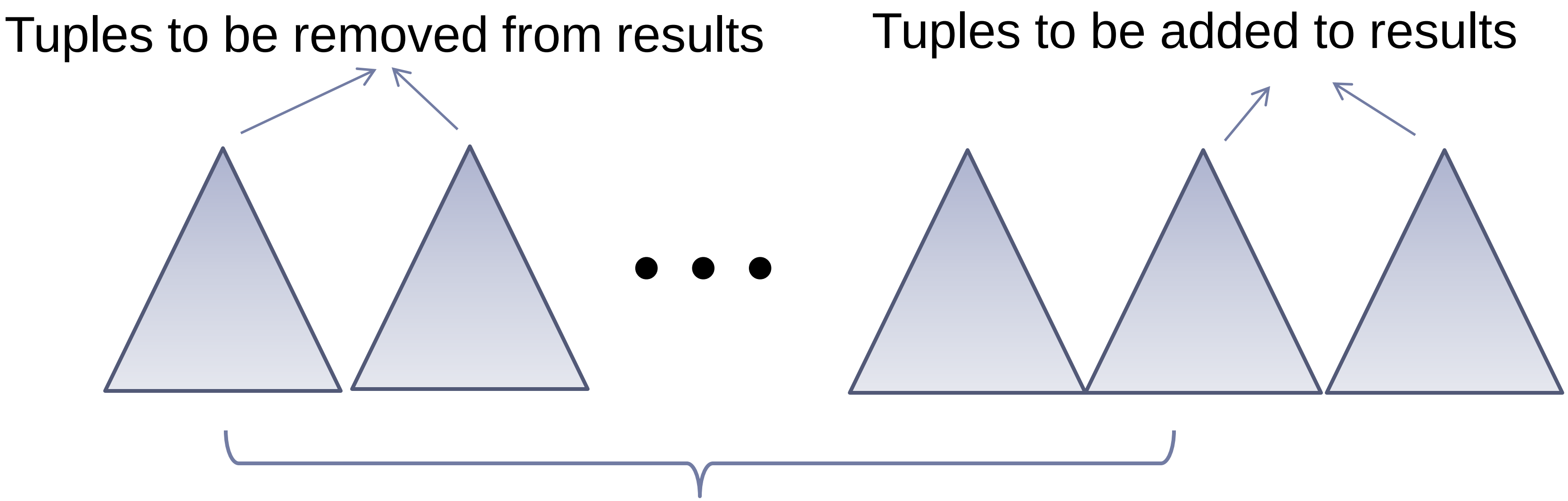
Time

t_i: timestamp of the *i*th tuple

TM-tree Continues



Sliding window query on the TM-tree

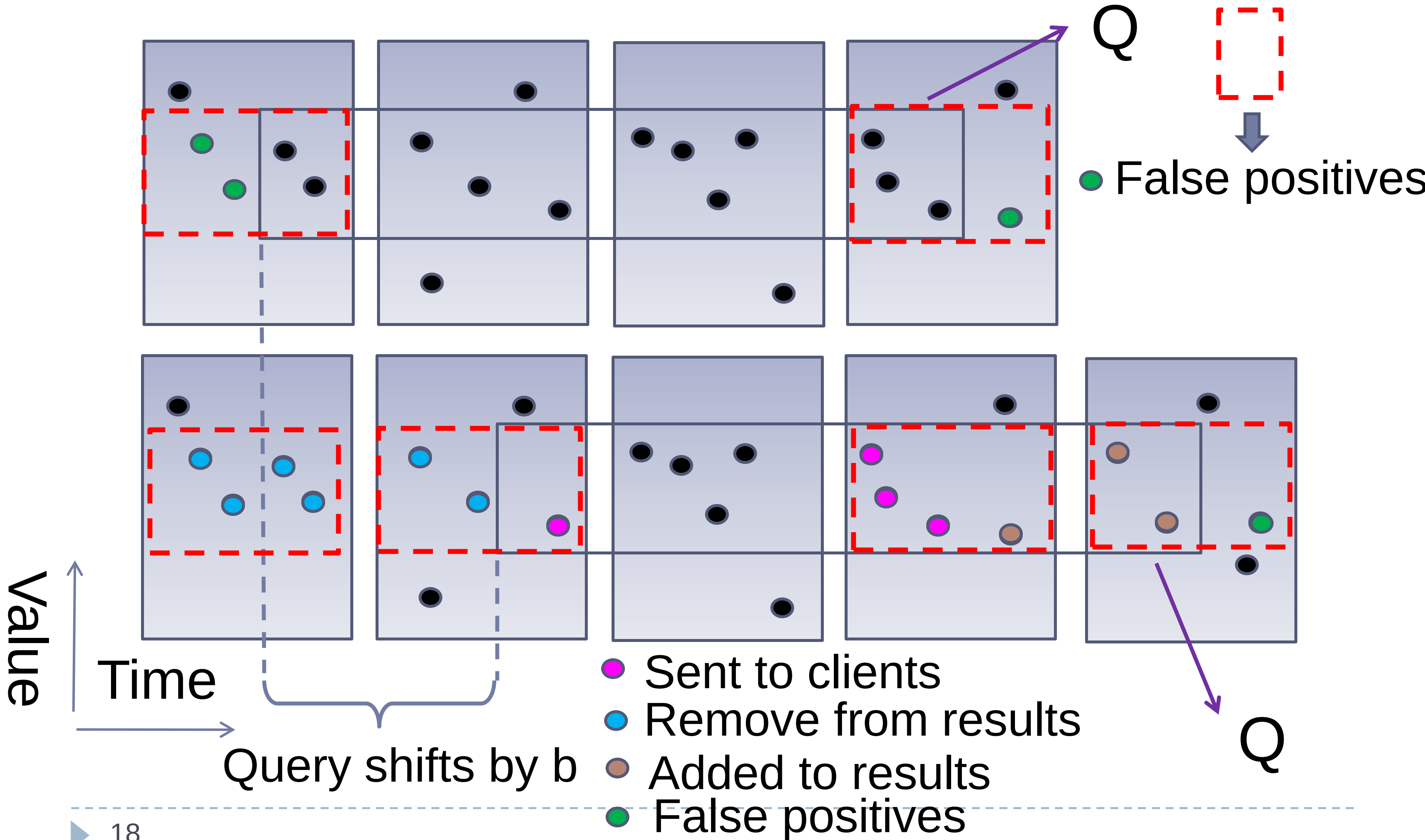


1. Initialization Query on trees

2. Window slides

3. Incremental update of query results on trees

Query the TM-tree



Correctness and Completeness

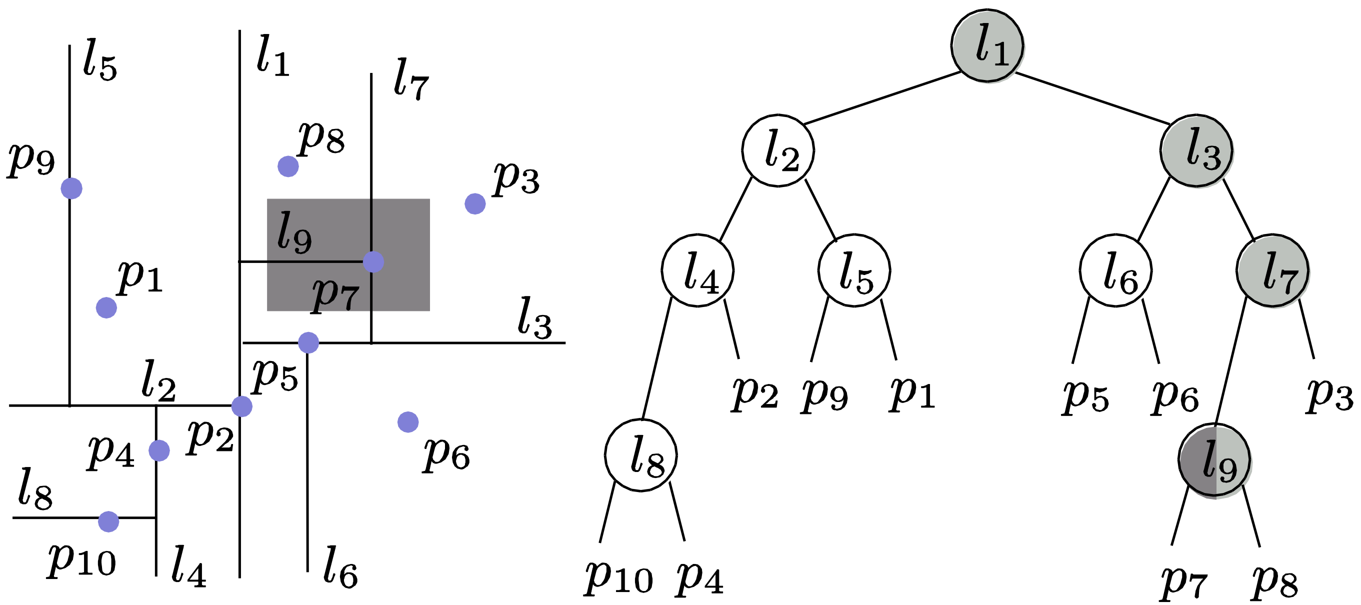
- Correctness:
  - Guaranteed by each individual Merkle tree
- Completeness:
  - Completeness in each small Merkle tree is guaranteed by what we have studied in the first part of this talk
  - Overall completeness:
    - Check that the results returned are obtained by querying consecutive trees that fall within the query range on time dimension and they completely cover the query range on time dimension.
    - This is possible as two boundary tuples' timestamps have been signed in each tree (hence these timestamps have to be included in the VO by the server).

Limitation of TM-tree

- Only supports one dimensional query
- False positives lead to large VO size, especially when each tuple has non-trivial size.

Merkle kd tree (Mkd-tree)

- To get rid of false positives:
  - Obviously we need a multi-dimensional indexing structure
  - KD-tree: an excellent candidate with bounded query performance of $O(\sqrt{b})$ and $O(b \log b)$ to bulk-load.
  - A space-partition structure: partition along each dimension in turn.



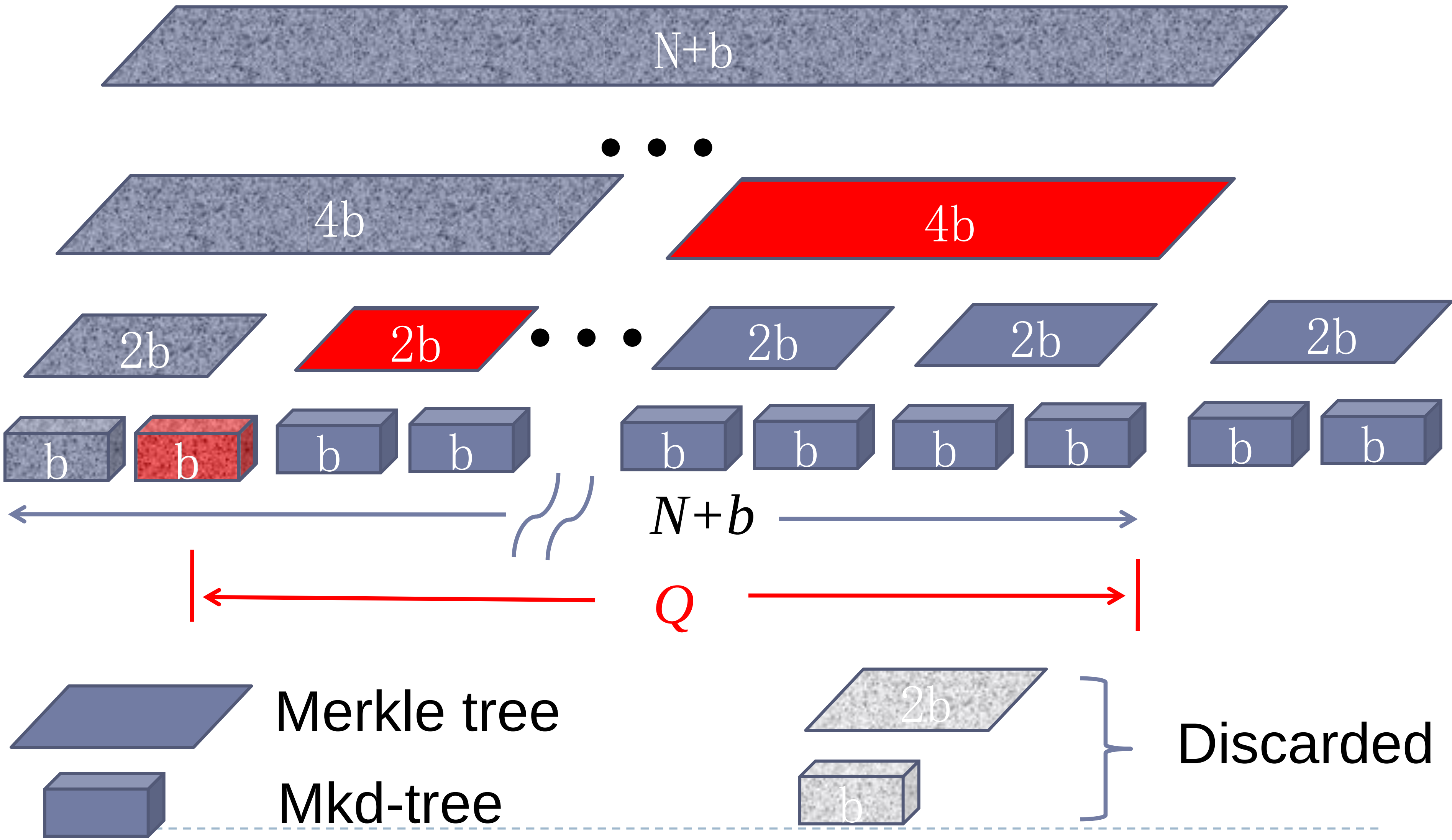
Mkd-tree and TMkd-tree

- Incorporating Merkle tree into KD-tree:
  - Leaf node: $H(p)$, p is the point contained in this node
  - Index node u with children v, w and dividing line l_u: $H(h_v|h_w|l_u)$
- Tumbling Merkle kd-tree (TMkd-tree)
  - Similar idea as it is in TM-tree, but we are using Mkd-tree as each small tree.
  - Boundary trees no longer introduce false positives!

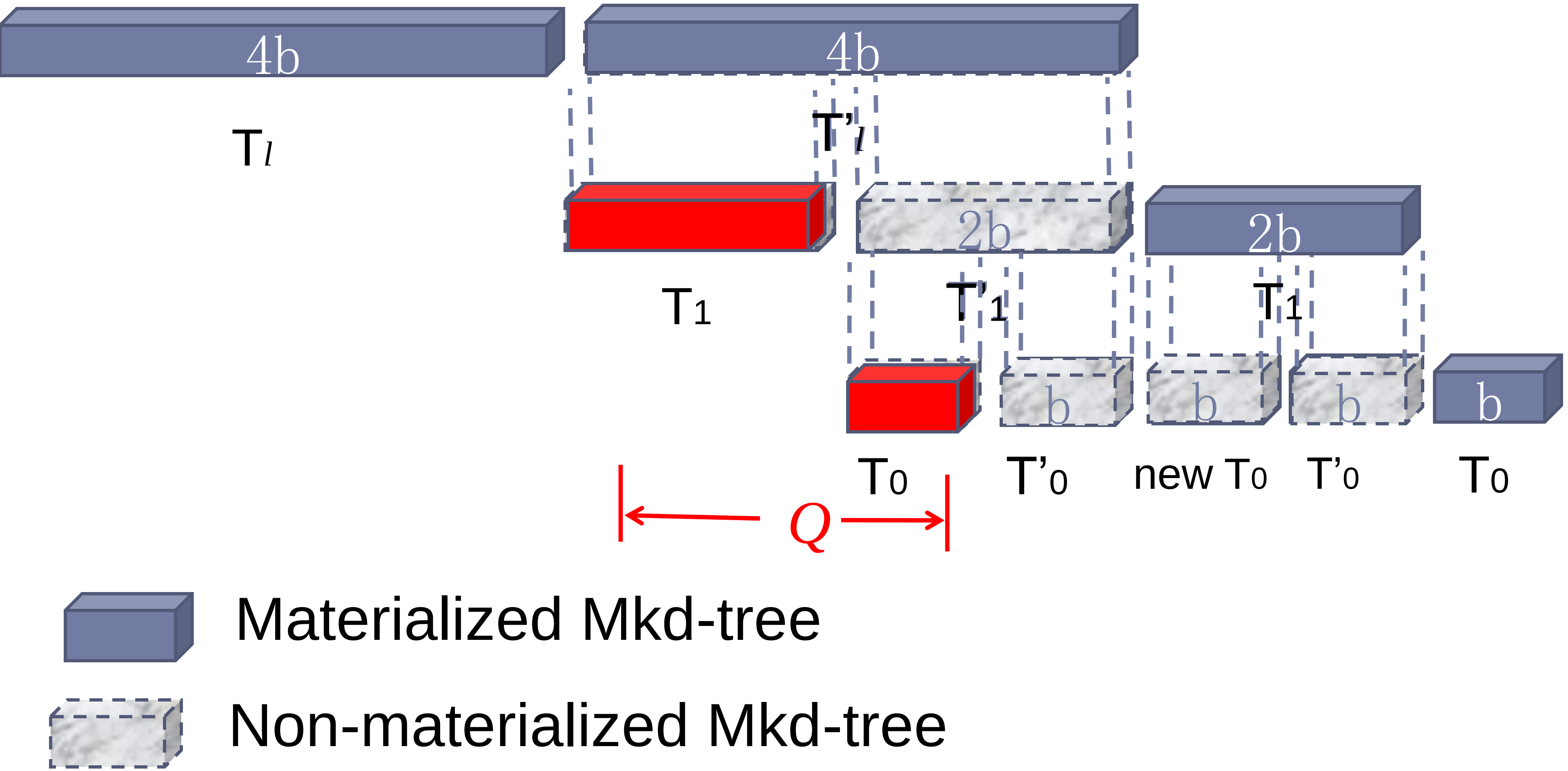
Is this good enough?

- Tumbling trees are good for maintaining the update to sliding window queries
- They both have linear space to N and $\log b$ update cost, and $\sigma/_b \log b + b + k$ or $\sigma/_b \sqrt{b} + k$ query cost
- But they are expensive for answering one-shot queries (or the initialization of sliding window queries)
  - query with window size n: have to query n/b trees: linear in n and could be expensive for large values of n.

Dyadic Merkle kd-tree (DMkd-tree): 1D queries



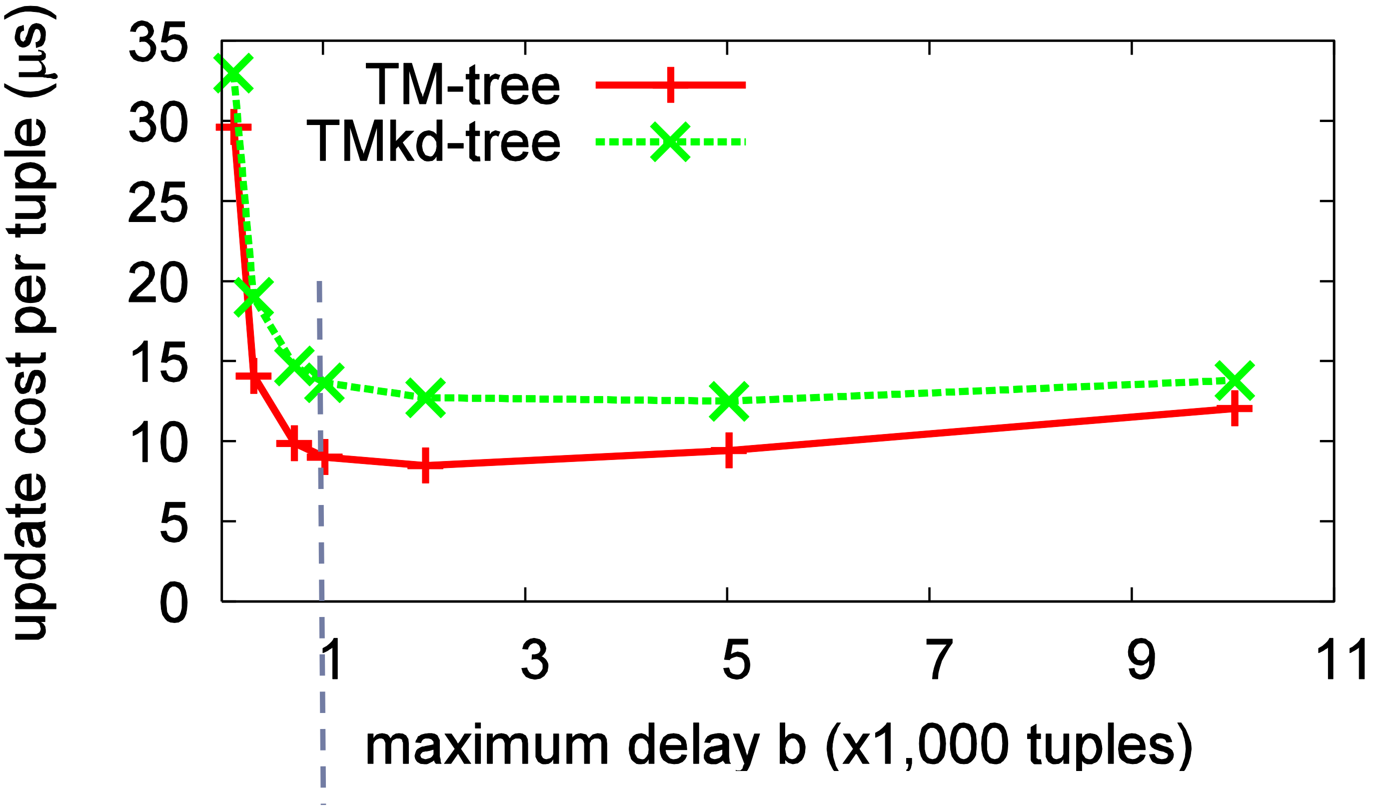
Exponential Merkle kd-tree (EMkd-tree):Multi-dimensional queries



Some Experiments

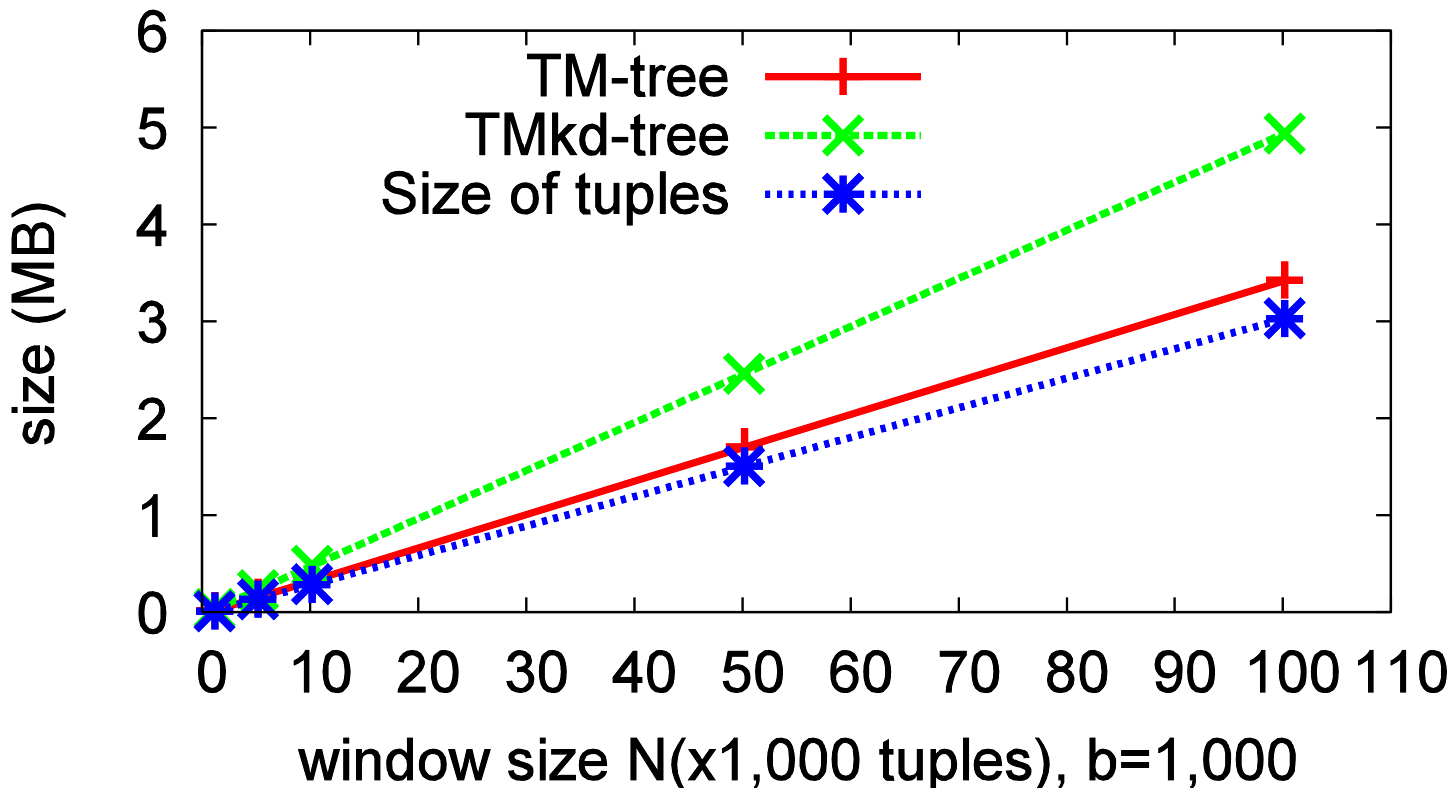
- We use real streams:
  - World Cup Data (WC)
  - IP traces from the AT&T network (IP)
- We perform the following query:
  - WC: Query attribute is the response size
  - IP: Query attribute is the packet size
- Hardware:
  - 2.8GHz Intel Pentium 4 CPU
  - Linux Machine

Tumbling trees: update cost



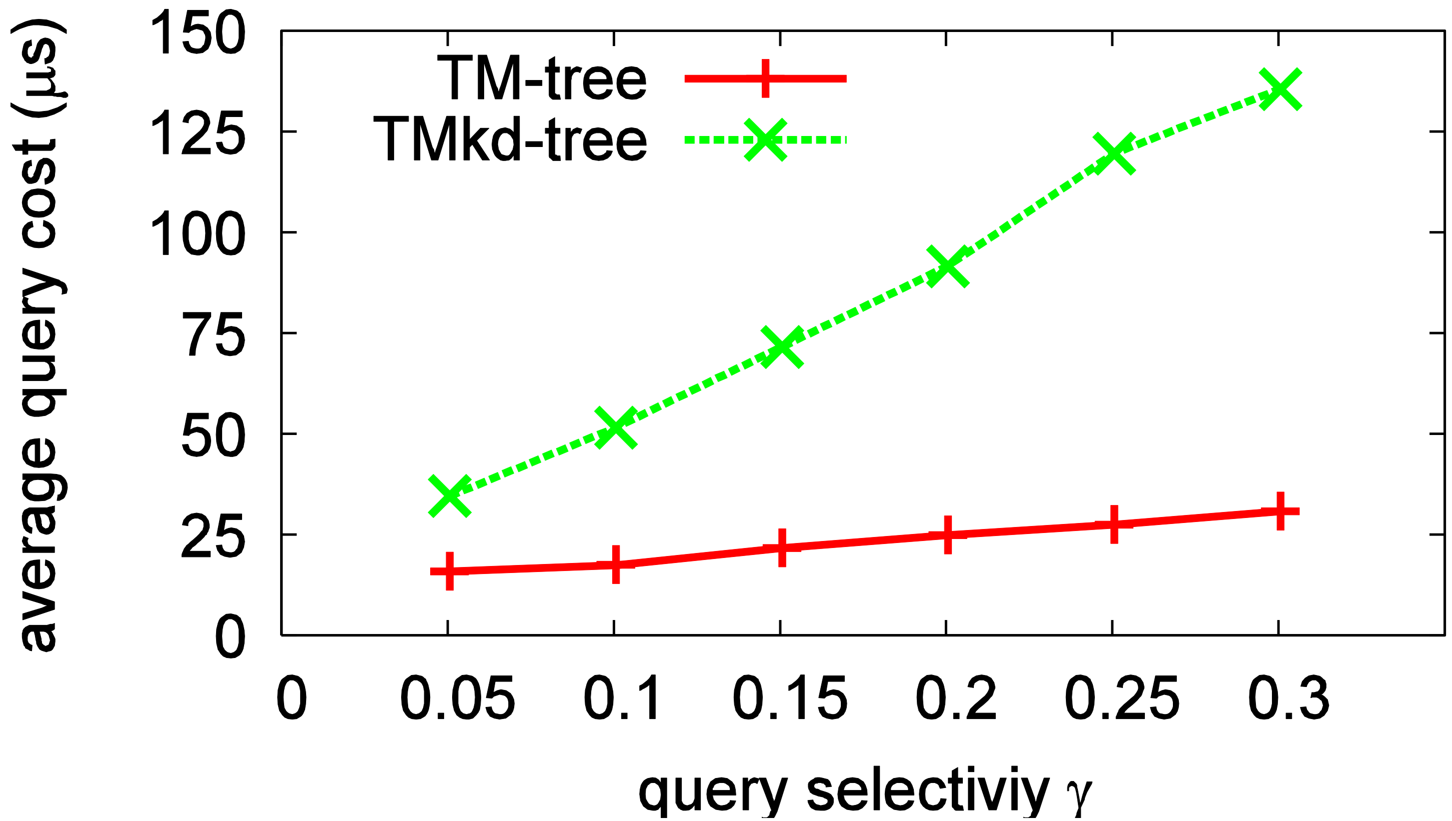
1. b=1000 is a sweet point
2. This delay is small: in real streams it spans less than one or two seconds

Tumbling trees: size



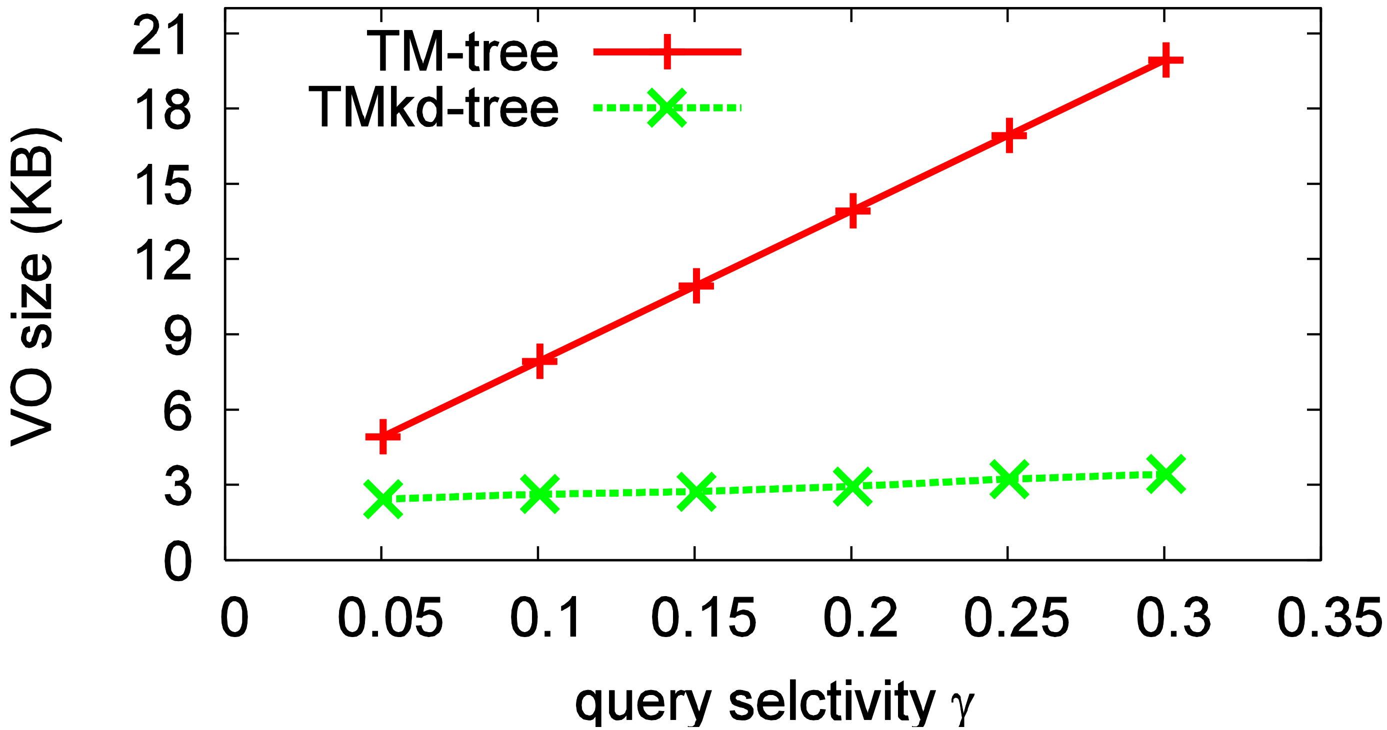
They both have linear size (to number of tuples covered in maximal window size of N)

Query cost per sliding period, b=1,000: fixed sliding period as b



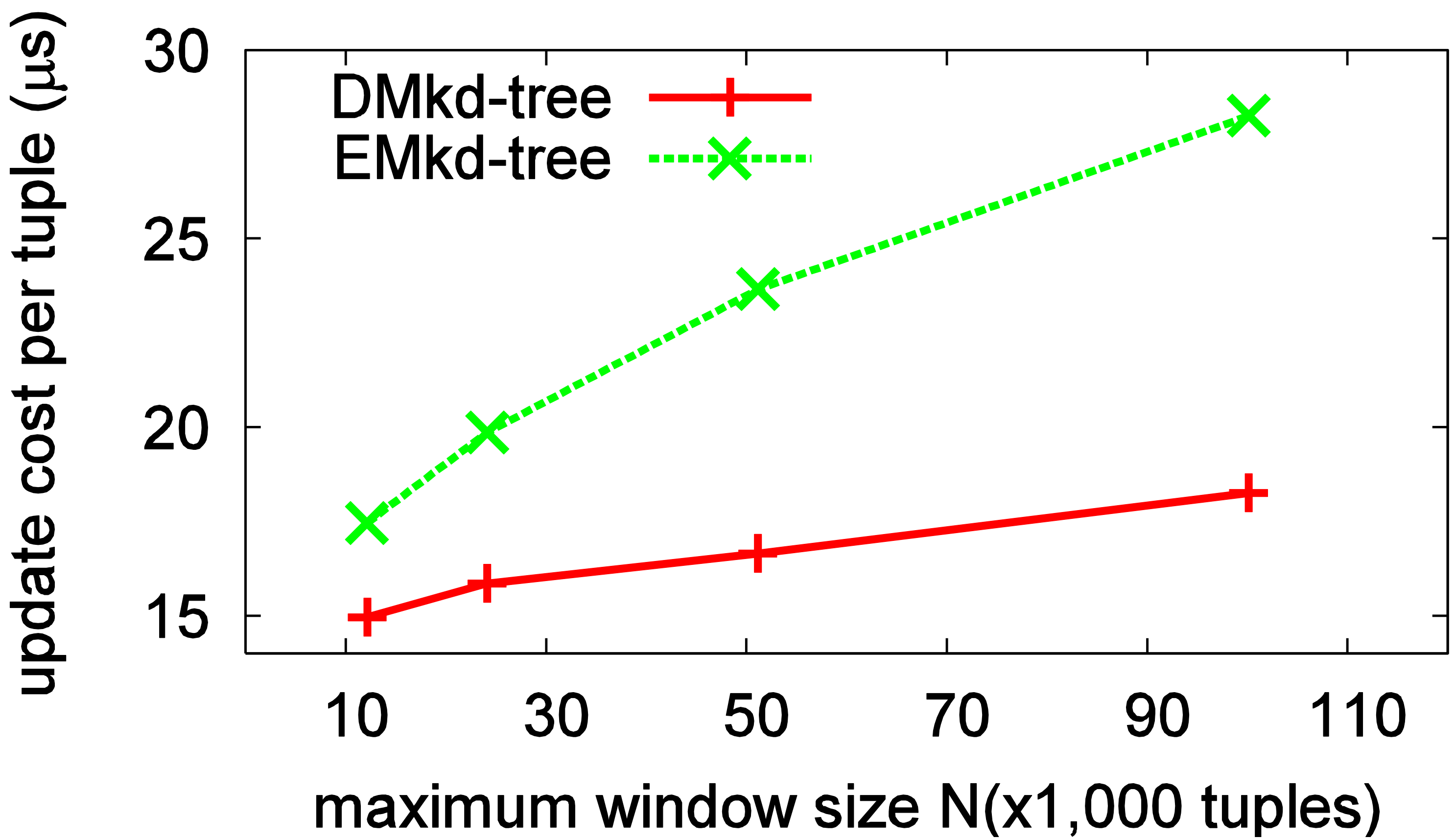
Linear scan of TM-tree at leaf level results in locality which greatly improves its performance

VO size per sliding period, b=1,000: fixed sliding period as b

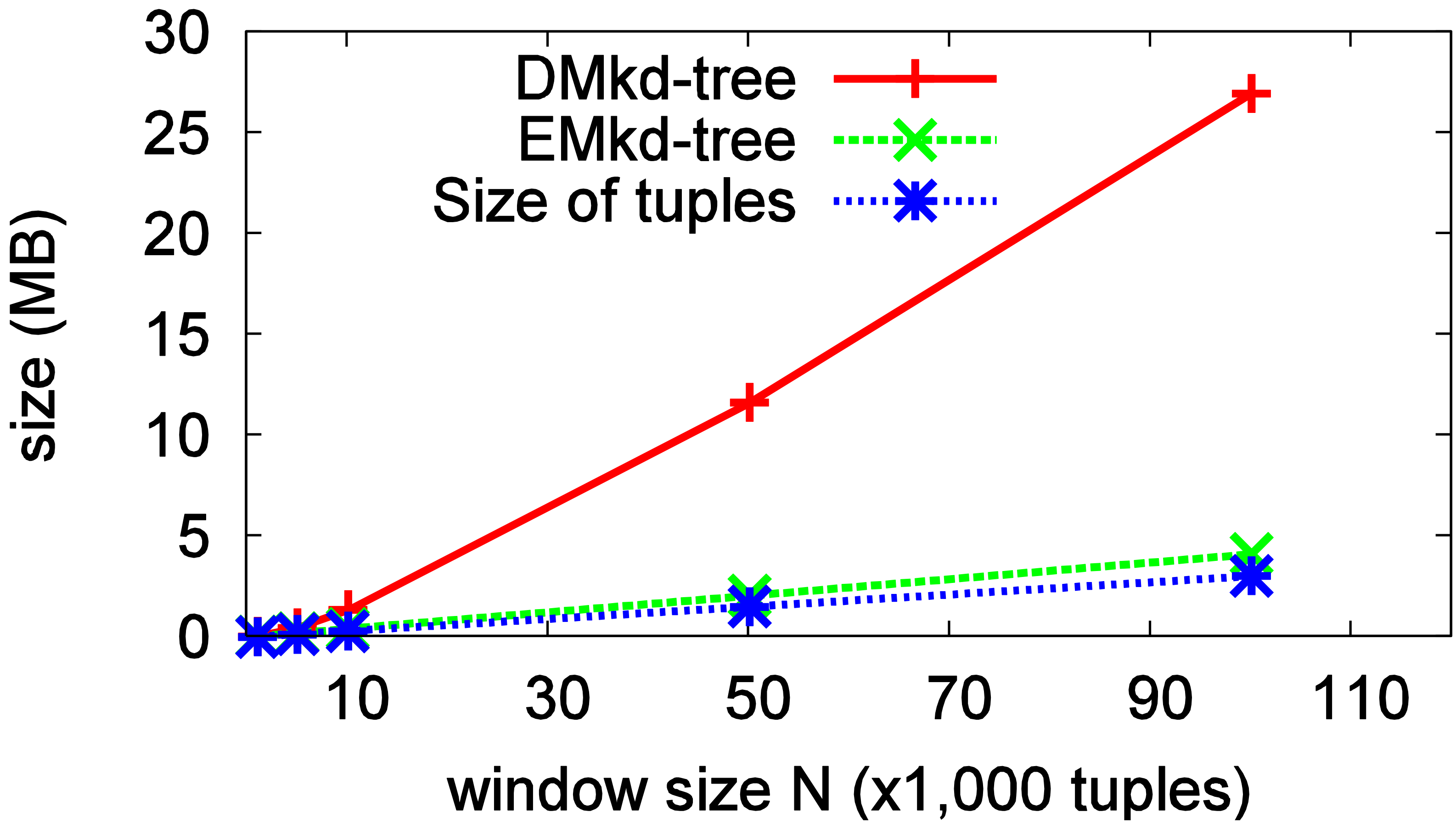


TM-tree incurs roughly 4γb false positives

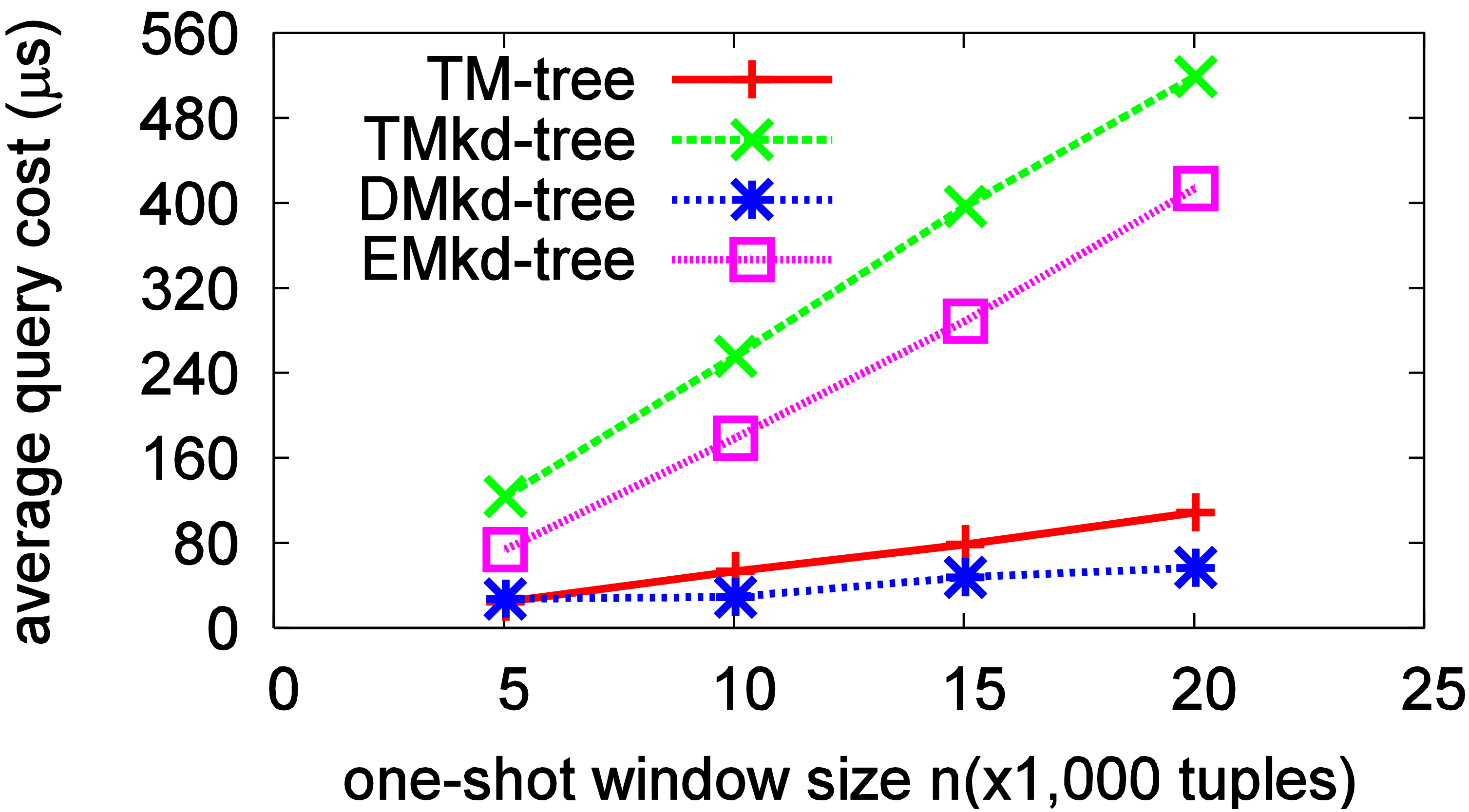
DM-kd Tree, EM-kd Tree Update Cost



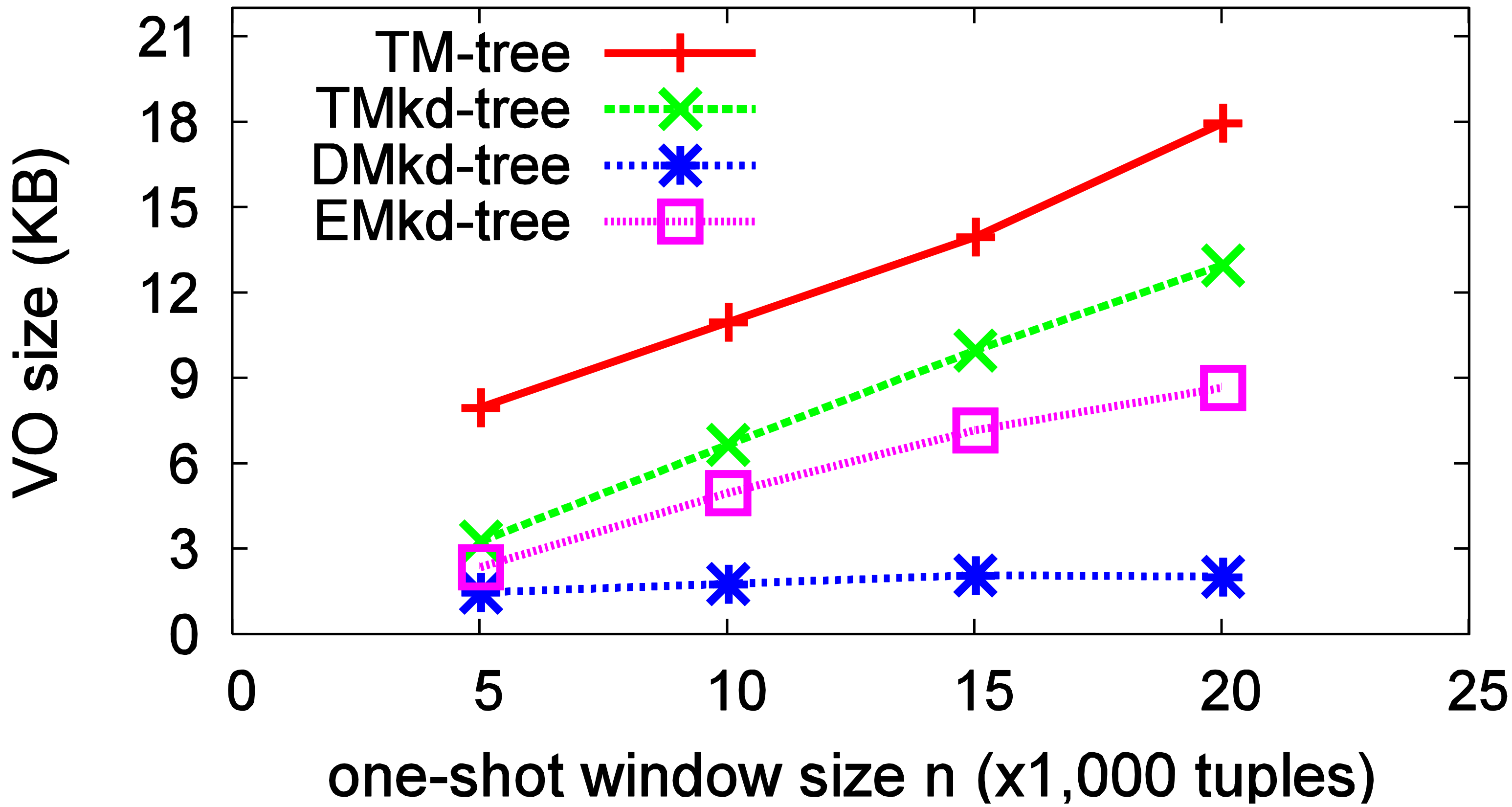
DMkd, EMkd trees: size



One Shot Query Cost



One Shot Query: VO size



Summary

- All trees support aagregations
- TM-tree and DMkd-tree support only one-dimensional queries
- TMkd-tree and EMkd-tree support multi-dimensional queries
- Tumbling trees are good for maintaining updates to sliding window queries, while DMkd-tree and Emkd-tree are good for one shot queries.

Thanks!

- Questions