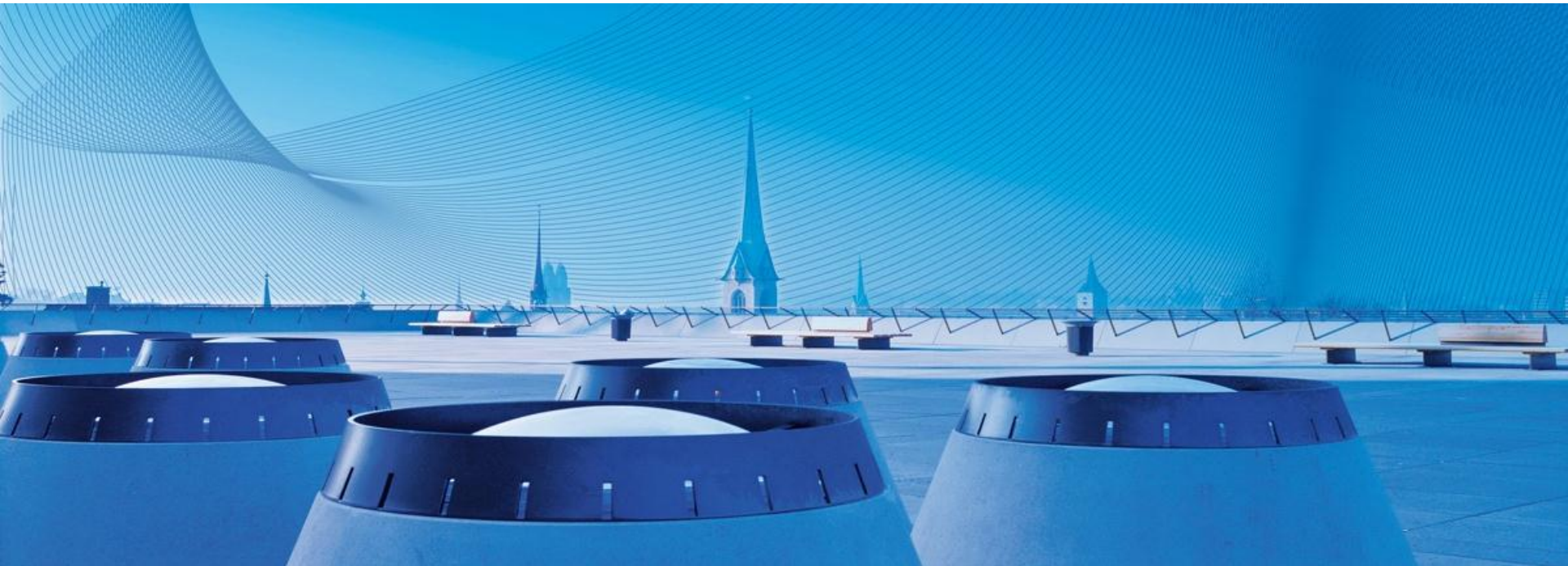


Indexing Moving Objects using Short-Lived Throwing Indexes

Advances in Spatial and Temporal Databases, 11th International Symposium, SSTD 2009, Aalborg, Denmark, July 8-10, 2009, Proceedings 2009
Jens Dittrich, Lukas Blunschi and Marcos Antonio Vaz Salles

A paper analysis by Willy Lai



Glossary

- MOVIES

MOVing objects Indexing using frEquent
Snapshots

Outline

- Introduction
 - Main Challenge
 - Optimizations
 - Mutlidimensional indexes
- MOVIES
 - Idea
 - Algorithm
 - Query processing
- Comparison
- Conclusion

Moving Objects in reality



Main Challenge



Optimizations

- Speed profile
 - Instead of location, store location vector
Reduces amount of updates up to a factor of 3
- Include environmental information
 - Cars usually do not leave the road network, we can restrict their possible locations to the network.

Multidimensional index structures

- R-Tree
- k-d-Tree
- Linearization (e.g. B^x-Tree)
- Other variations seen today

Multidimensional index structures

- R-Tree
- k-d-Tree
- Linearization (e.g. B^x-Tree)
- Other variations seen today

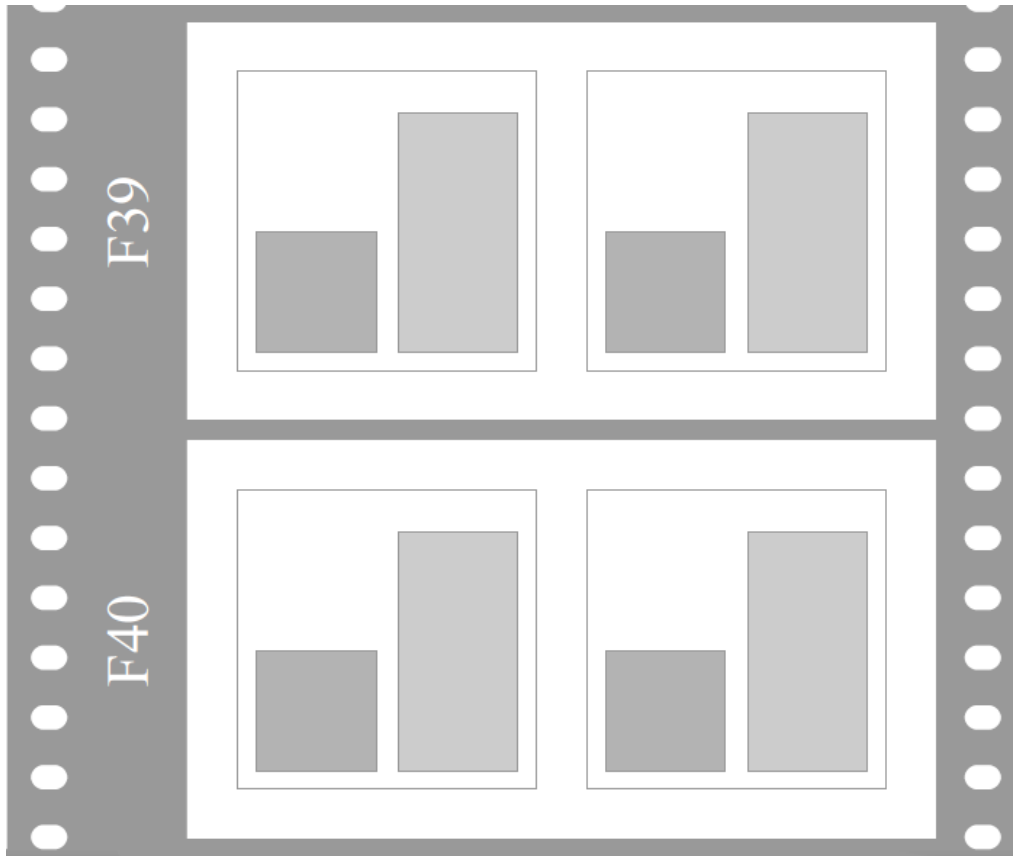
Created once, kept forever and apply updates immediately.

MOVIES

MOVIES Idea

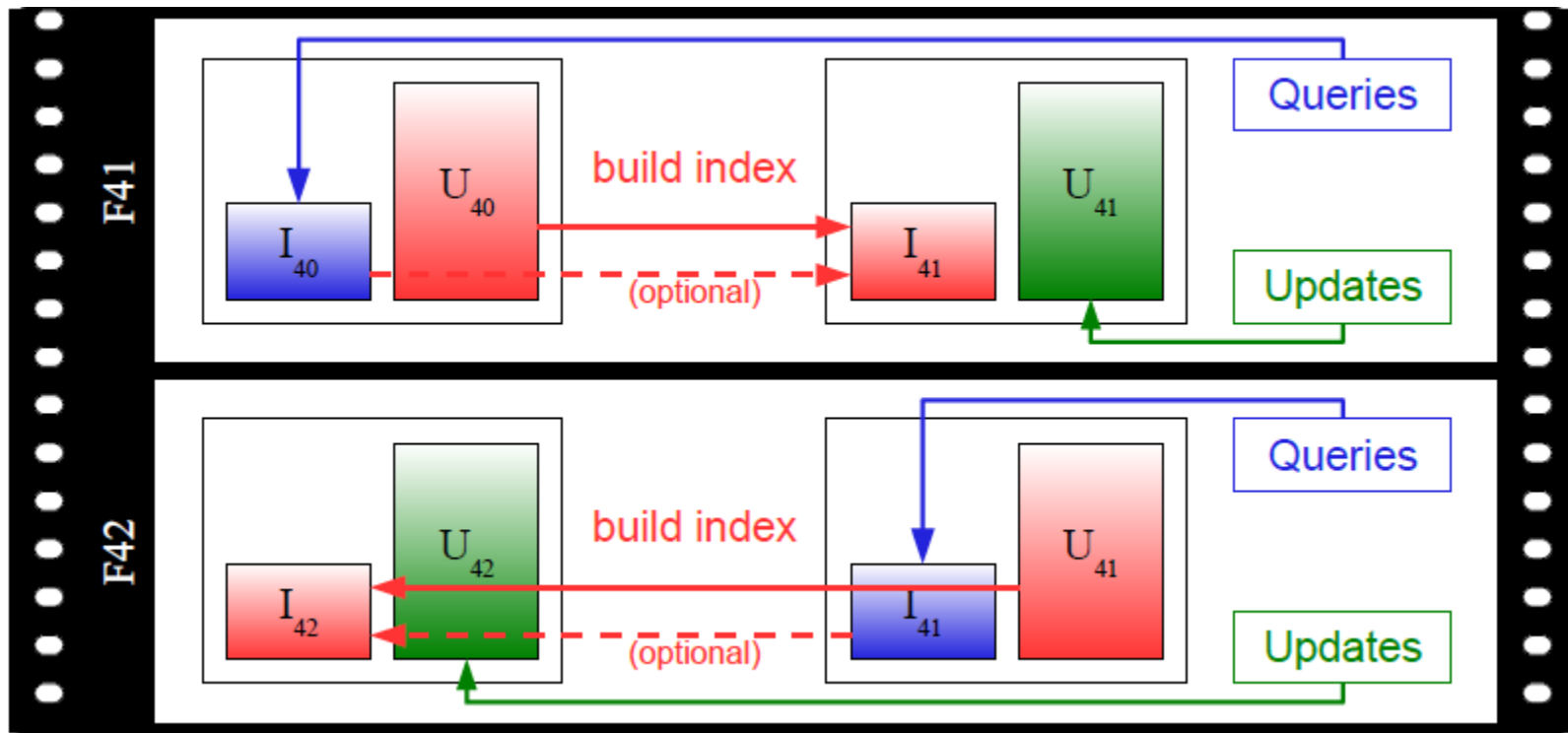
- Most application can tolerate some staleness in their data
- Do not apply updates immediately
- Buffer and rebuild

MOVIES Algorithm



- Collect arriving updates in a buffer
- Answer queries by traversing the index built in the previous frame
- Build a new index from the previous buffer

MOVIES Algorithm



MOVIES Data structures

All datastructures are held completely in main memory

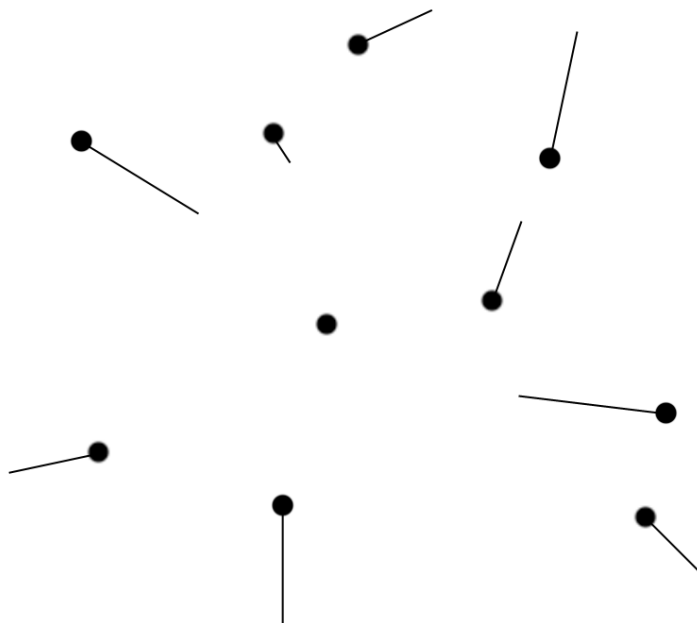
- Update buffers
 - Logged buffers
 - Aggregation buffers
- Indexes
 - Read-only
 - Linearized
 - Stored in an array

MOVIES Query processing

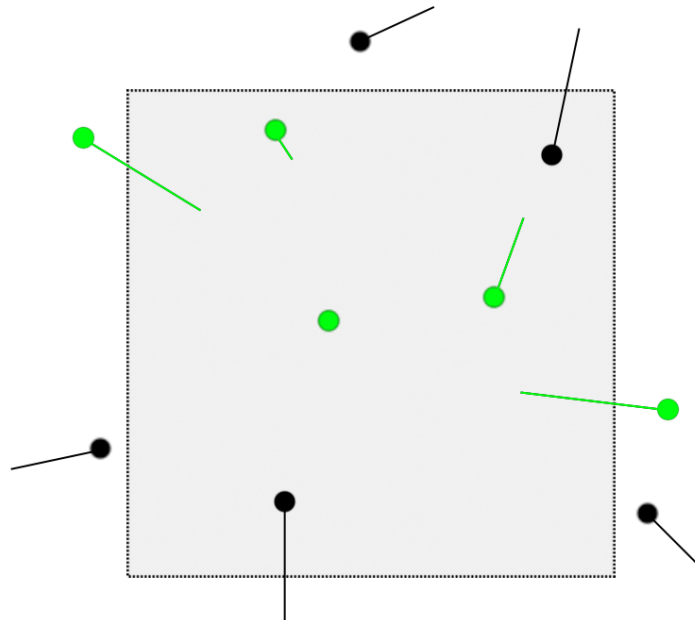
The queries are time parameterized range queries

- Predictive MOVIES
 - Only one index time
 - Space efficient
- Non-Predictive MOVIES
 - Each entry has its own timestamp
 - Update efficient

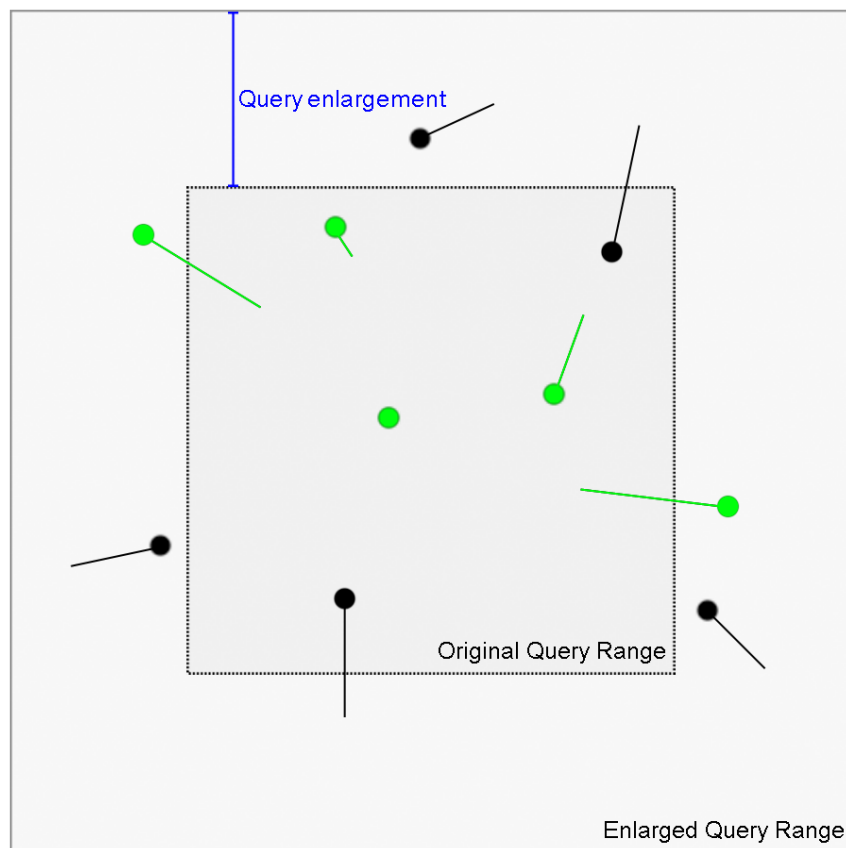
Query enlargement



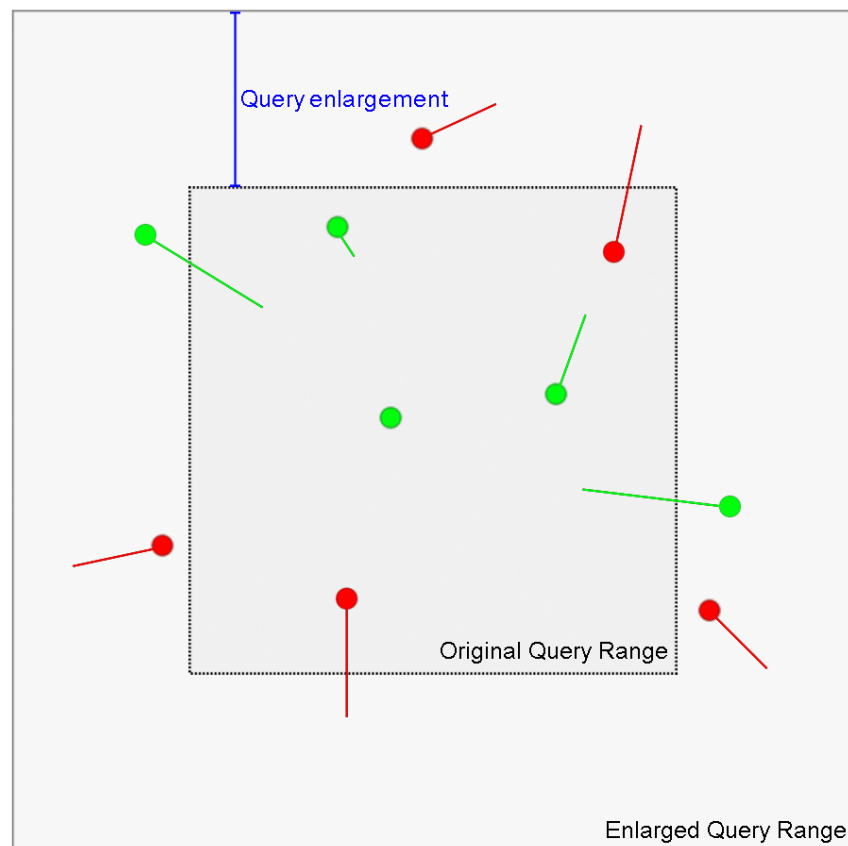
Query enlargement



Query enlargement



Query enlargement



Comparison

Update in place

- Assume one dimensional index structure (e.g. binary tree)
- An update of an index involves in worst case
 - 1 deletion of a old node: $O(\log(N))$
 - 1 insertion of a new node: $O(\log(N))$
- Costs
 - N: number of entries
 - a: system dependent constant
 - Cost for k updates: $2 \cdot k \cdot a \cdot \log(N)$

MOVIES Costs

- Assume one dimensional index structure (e.g. binary tree)
- Collect and rebuild involves
 - Insertion into buffer $O(1)$
 - Bulkloading index $O(N \cdot \log(N))$
- Costs
 - N : number of entries
 - b, c : system dependent constants
 - Cost of bulkloading k entries: $k \cdot b + c \cdot N \cdot \log(N)$

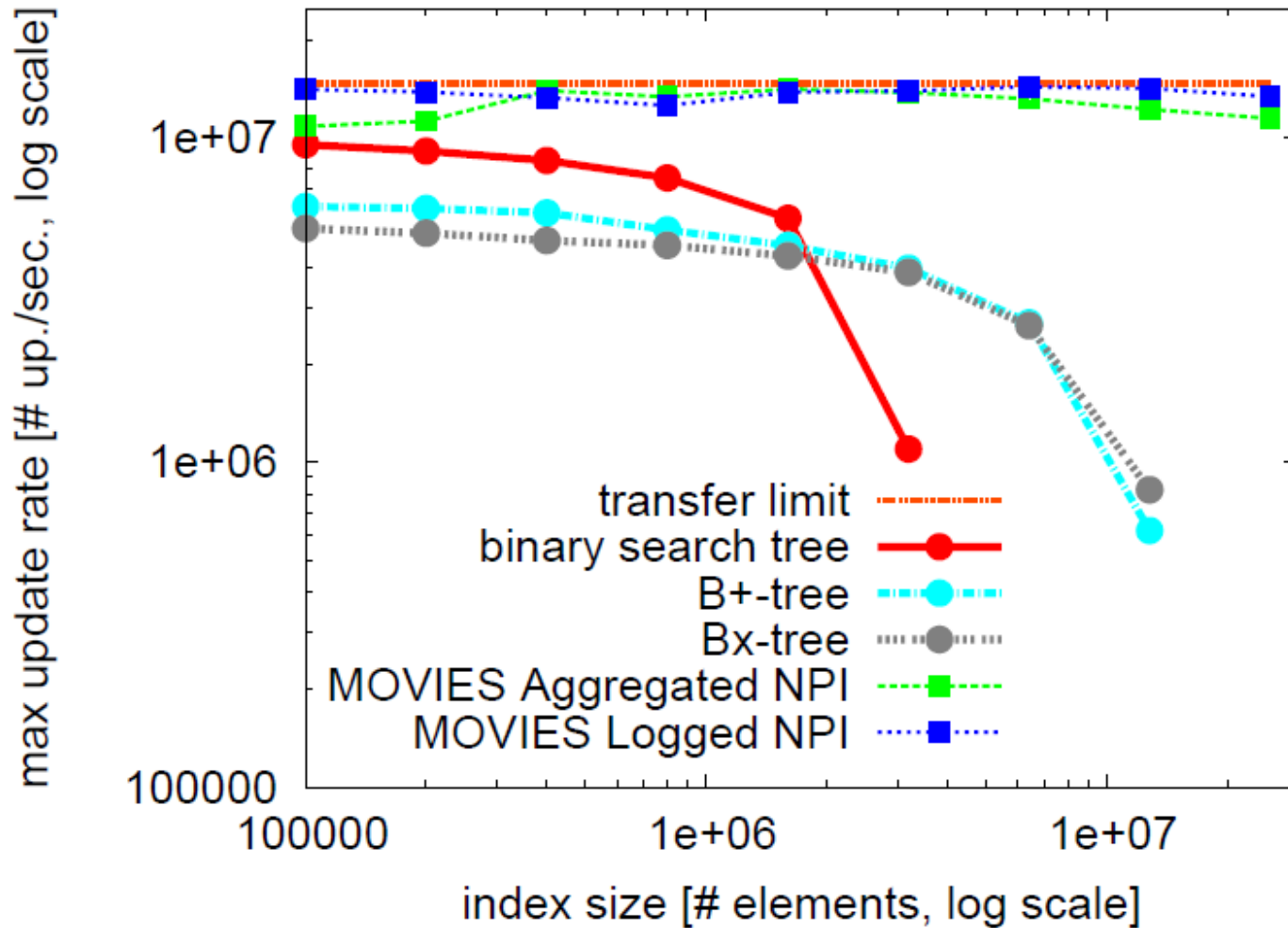
In Theory

- When will MOVIES perform better than Update-in-place?

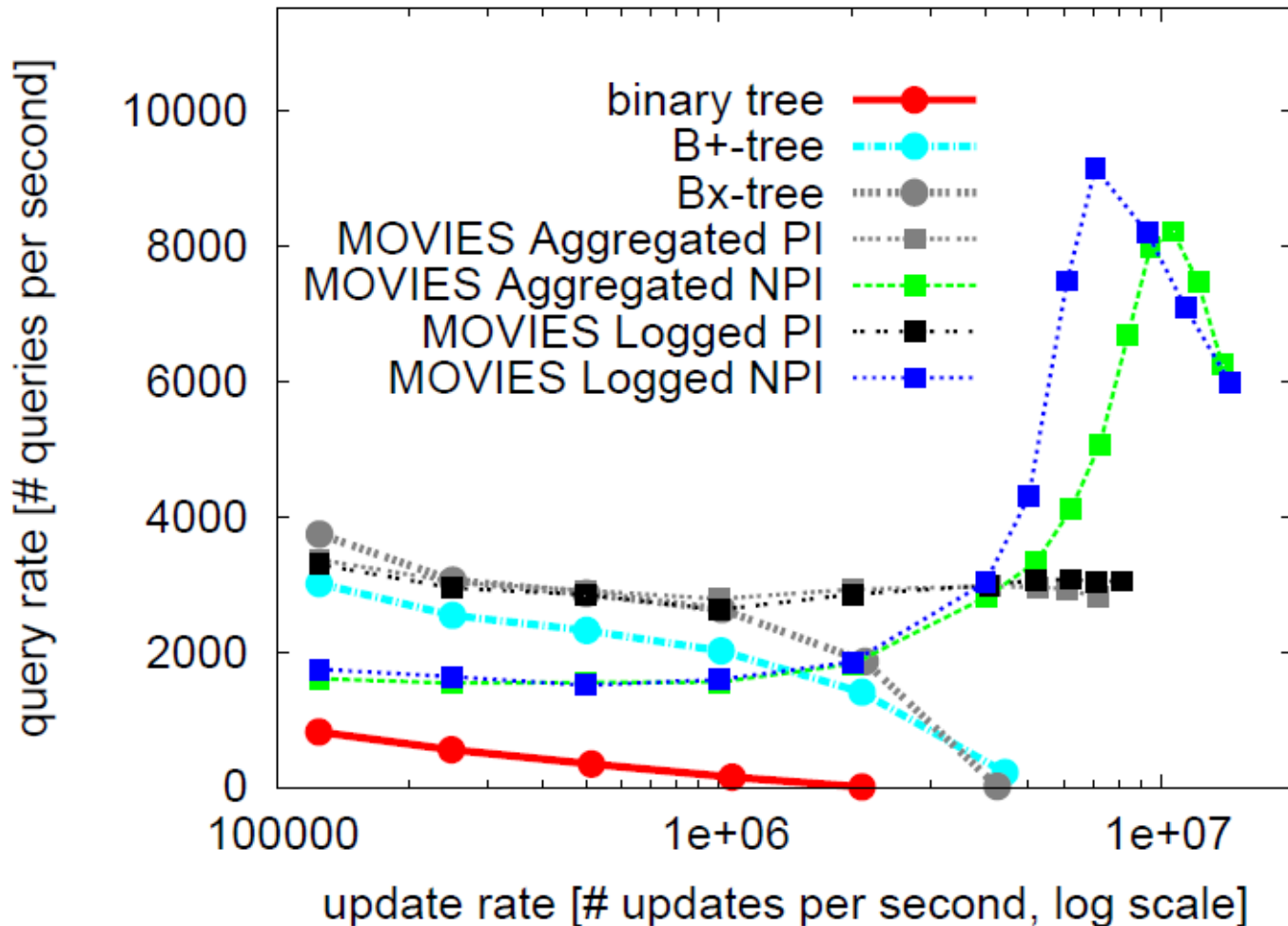
$$2 \cdot k \cdot a \cdot \log(N) > k \cdot b + c \cdot N \cdot \log(N)$$

- Estimated upper bounds:
 - $a = 73.6 \text{ ns}$
 - $b = 8.9 \text{ ns}$
 - $c = 112.5 \text{ ns}$
- For an index size $N = 1'000'000$
When $k = 62'872$ MOVIES performs better than Update-in-place

In Practice



In Practice



Conclusion

- Constants do matter!
- MOVIES
 - Staleness –
 - Update performance ++
 - Query performance ++