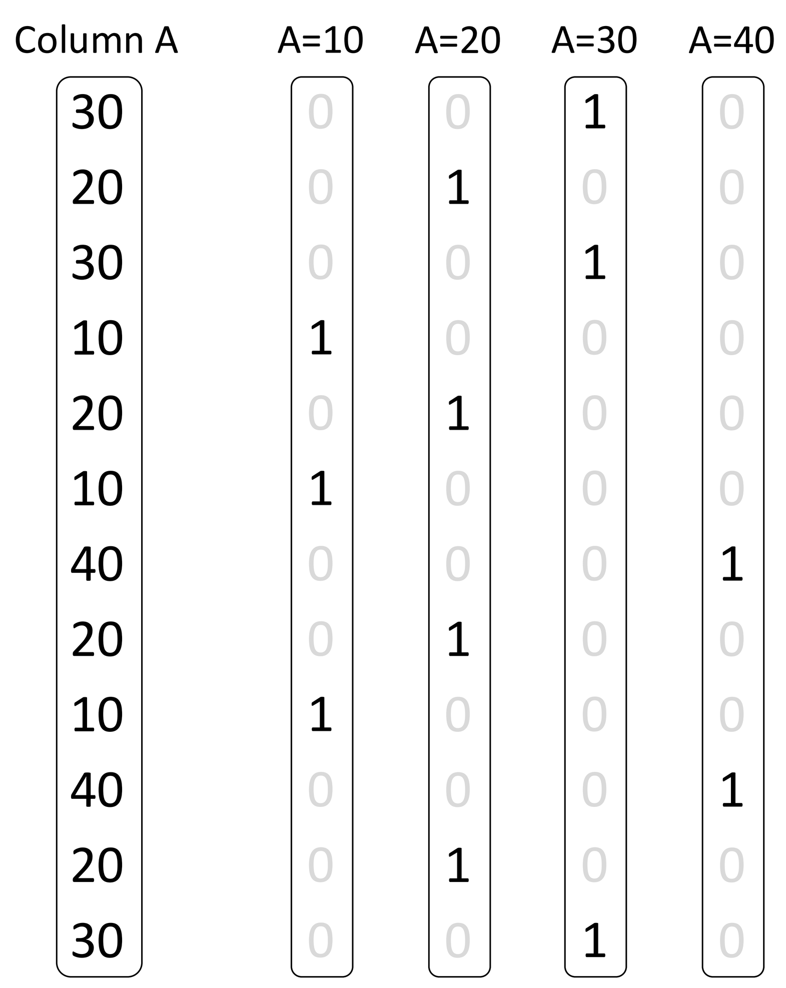
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

UpBit: Scalable In-Memory Updatable Bitmap Indexing

Manos Athanassoulis, Zheng Yan, Stratos Idreos

Indexing for Analytical Workloads

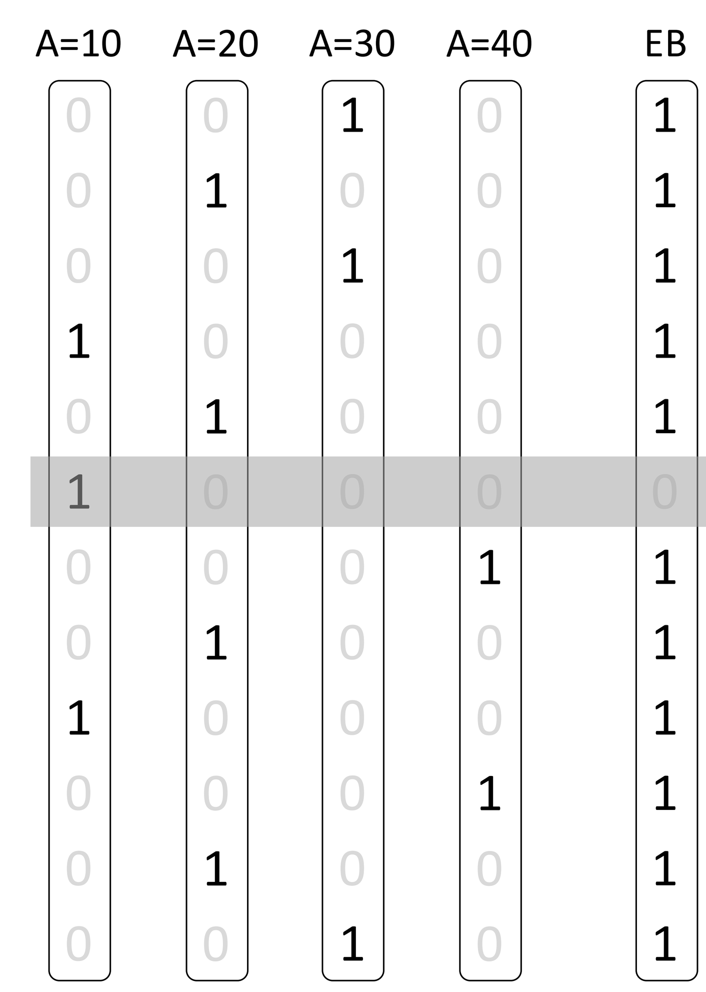


- ✓ Compact representation of query result
- ✓ Query result is readily available

Bitvectors
- ✓ Can leverage fast Boolean operators
- ✓ Bitwise operations over bitvectors
- ✓ More space-efficient for low domain cardinality
- ✗ Space-inefficient for larger domains
  - ✓ Addressed by bitvector encoding/compression
  - **core idea**: *run-length encoding* in prior work
- ✗ Updating encoded bitvectors is **very** inefficient

Goal: Bitmap Indexing for Reads & Updates

Existence Bitvector is the bottleneck

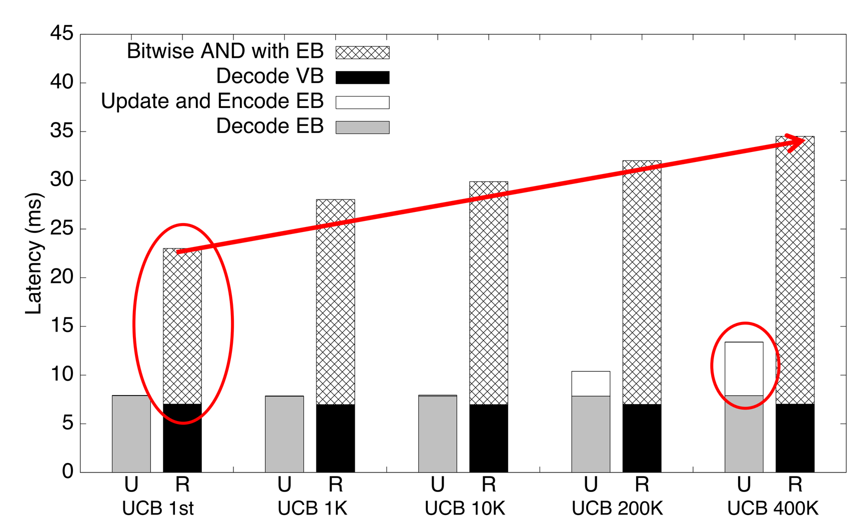


Update Conscious Bitmaps (UCB), SSDBM 2007

deletes by invalidation

reads: bitwise AND with EB

updates: delete-then-append

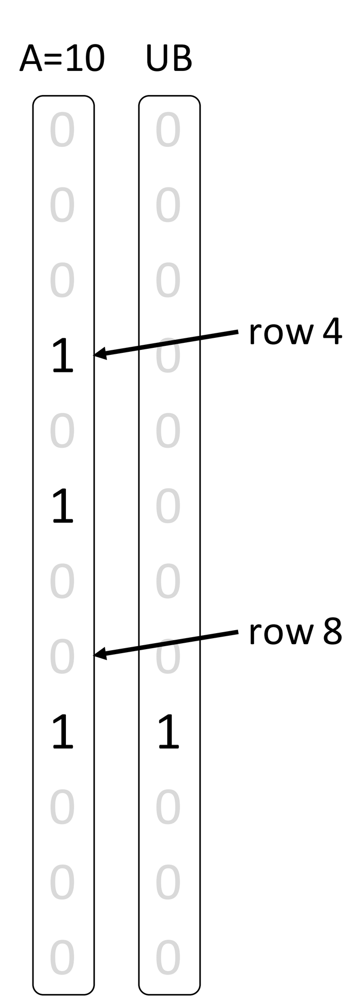


Challenge: read cost increases with updates

UpBit Design & Operations

Bottleneck 1: single auxiliary bitvector
- **Design Element 1**: multiple auxiliary bitvectors to distribute the cost
- **Design Element 2**: efficient random access in bitvectors

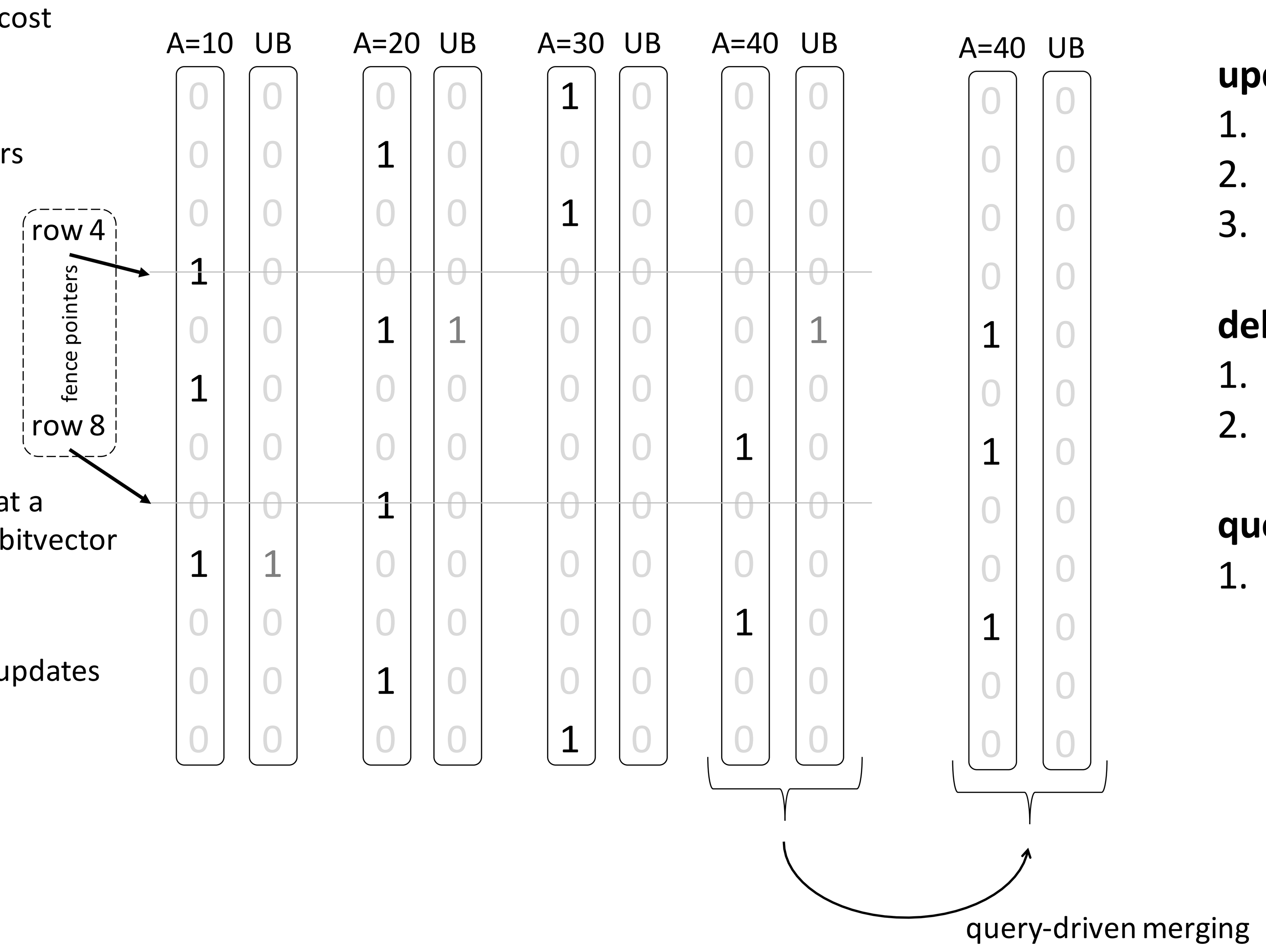
Bottleneck 2: repetitive bitwise operations
- **Design Element 3**: query-driven merging of aux to main bitvectors



Design Element 1: update bitvectors
initialized to 0s
current value is the XOR
every update flips a bit

Design Element 2: fence pointers
allow access to the *compressed word* at a known position of the *uncompressed* bitvector

Design Element 3: query-driven merging
use past bitwise operations to merge updates



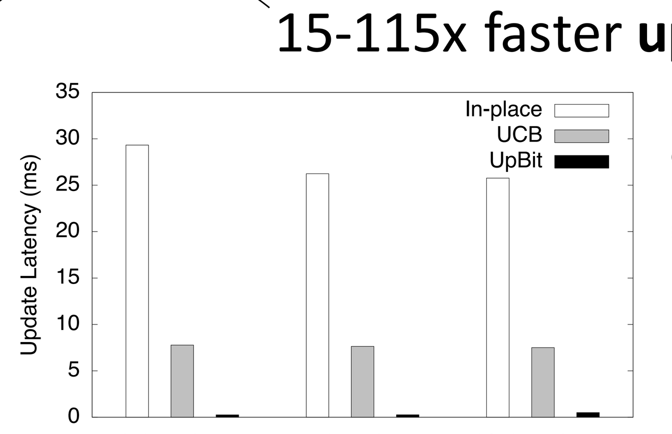
update row 5 to 40
1. find old value (20)
2. flip UB of row 5 of A=20
3. flip UB of row 5 of A=40

delete row 9
1. find old value (10)
2. flip UB of row 9 of A=10

query A=40
1. return XOR of A=40 and UB

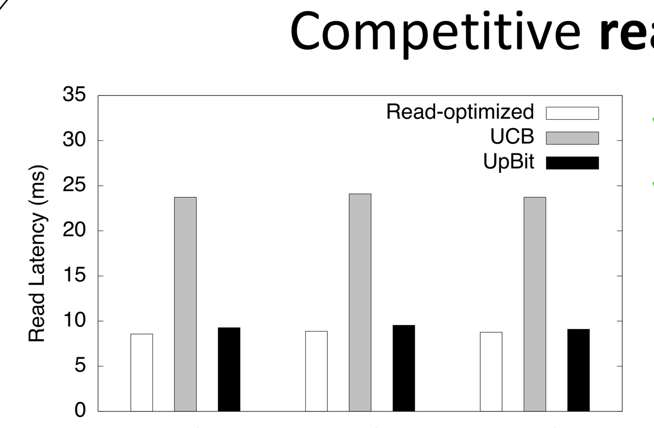
Scalable Updates and Efficient Reads with UpBit

15-115x faster **updates**



n=100M tuples,
d=100 domain values,
100k queries
(varying % of updates)

Competitive **reads**

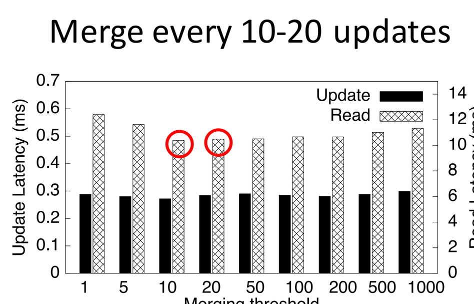


- ✓ 3x faster than UCB
- ✓ 8% overhead over read-optimized

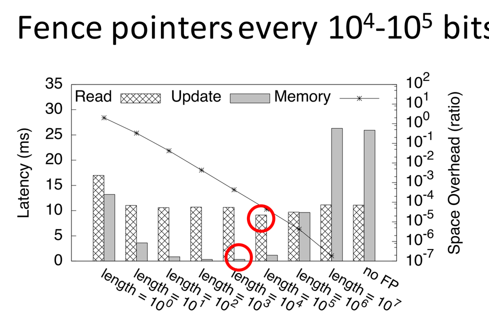
More in the paper …
- ✓ Multi-threaded support for reads and updates
- ✓ Impact of data skew
- ✓ Experiments with scientific data
- ✓ Full query analysis (including TPCH)

Tuning: merging and fence pointers

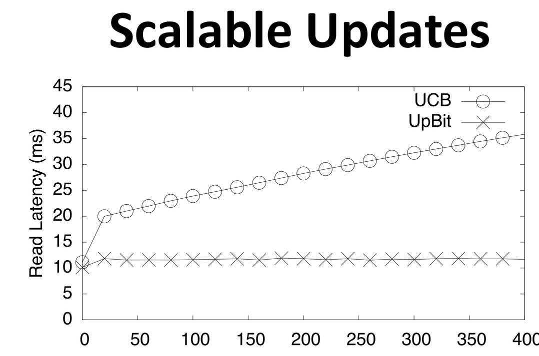
Merge every 10-20 updates



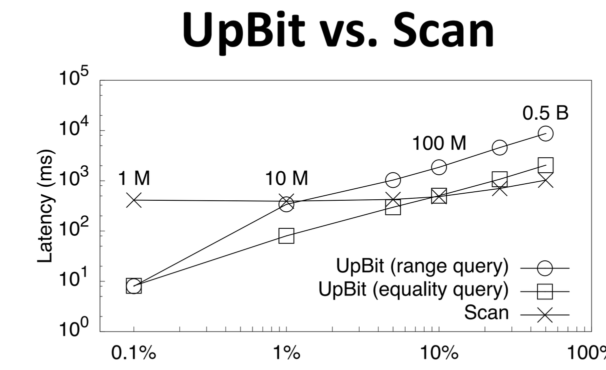
Fence pointers every 10^4-10^5 bits



Scalable Updates



UpBit vs. Scan



n = 1B, d = 1000 distinct domain values (range)
n = 1B, d vary for equality: 1000,100,10,1

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

daslab.seas.harvard.edu

DASlab