MTASet: A Tree-based Set for Efficient Range Queries in Update-heavy Workloads

Daniel Manor, Mor Perry, Moshe Sulamy

The Academic College of Tel Aviv-Yaffo

- A concurrent set data structure specifically optimized for environments characterized by high update throughput and frequent range queries
- Based on an (a, b)-tree
- Relaxed balancing
- Lock based
- Optimistic concurrency control
- Leveraging a tailored multi-versioning
- Supports **find(k), insert(k, v), delete(k)** and **scan(fromKey, toKey)**
- Range queries are wait-free
- Linearizable

Three types of nodes: leaf nodes, internal nodes and tagged internal nodes.

Internal nodes are used for routing

Tagged internal nodes indicate a temporary height imbalance in the tree, resulting from relaxed balancing.

Leaf nodes are the only nodes that store values

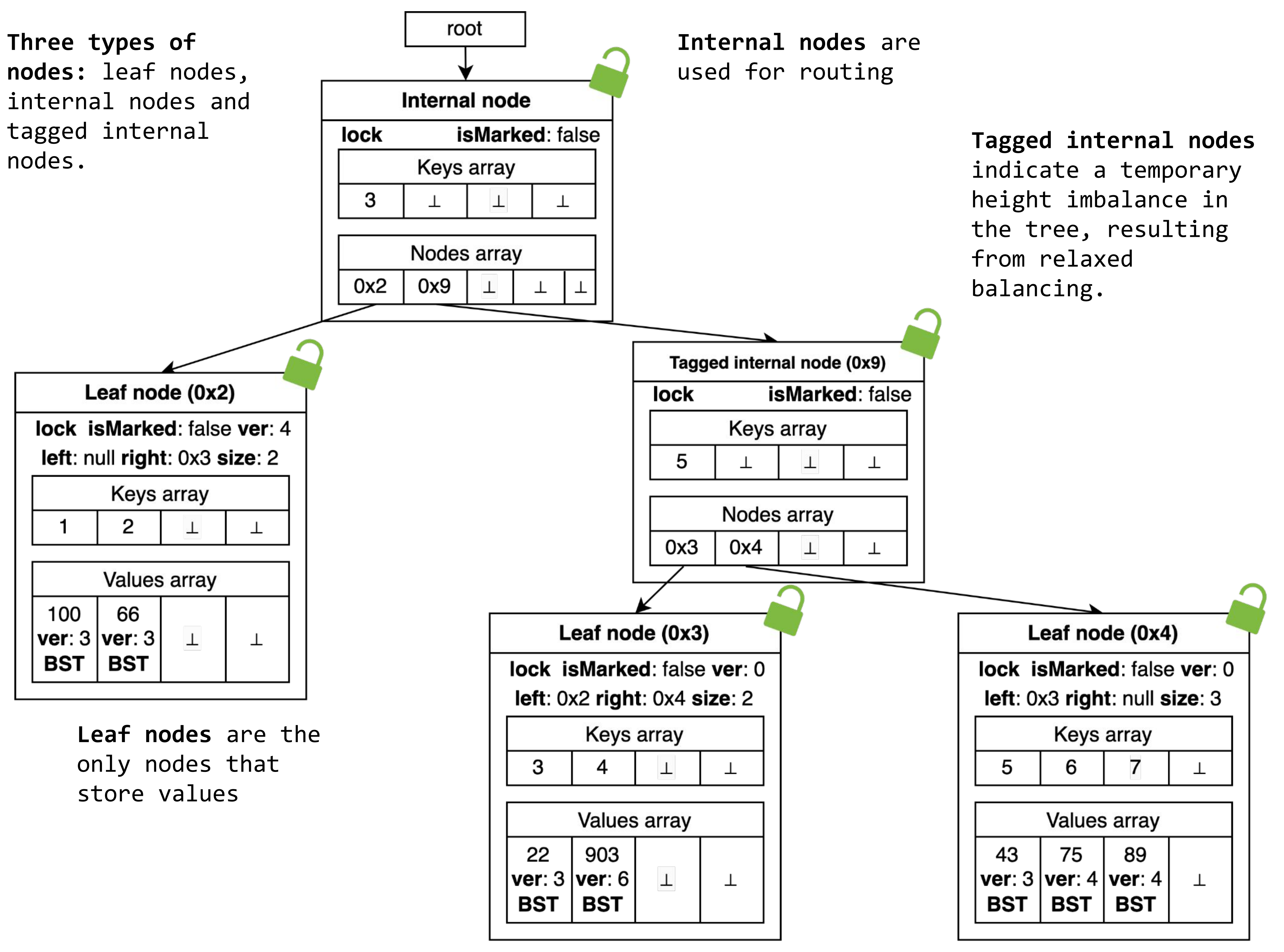


Figure 2. A snapshot of MTASet, a=2, b=4. An internal node pointing to a tagged internal node and a leaf node. The tagged internal node points to two leaf nodes. No locks are acquired.

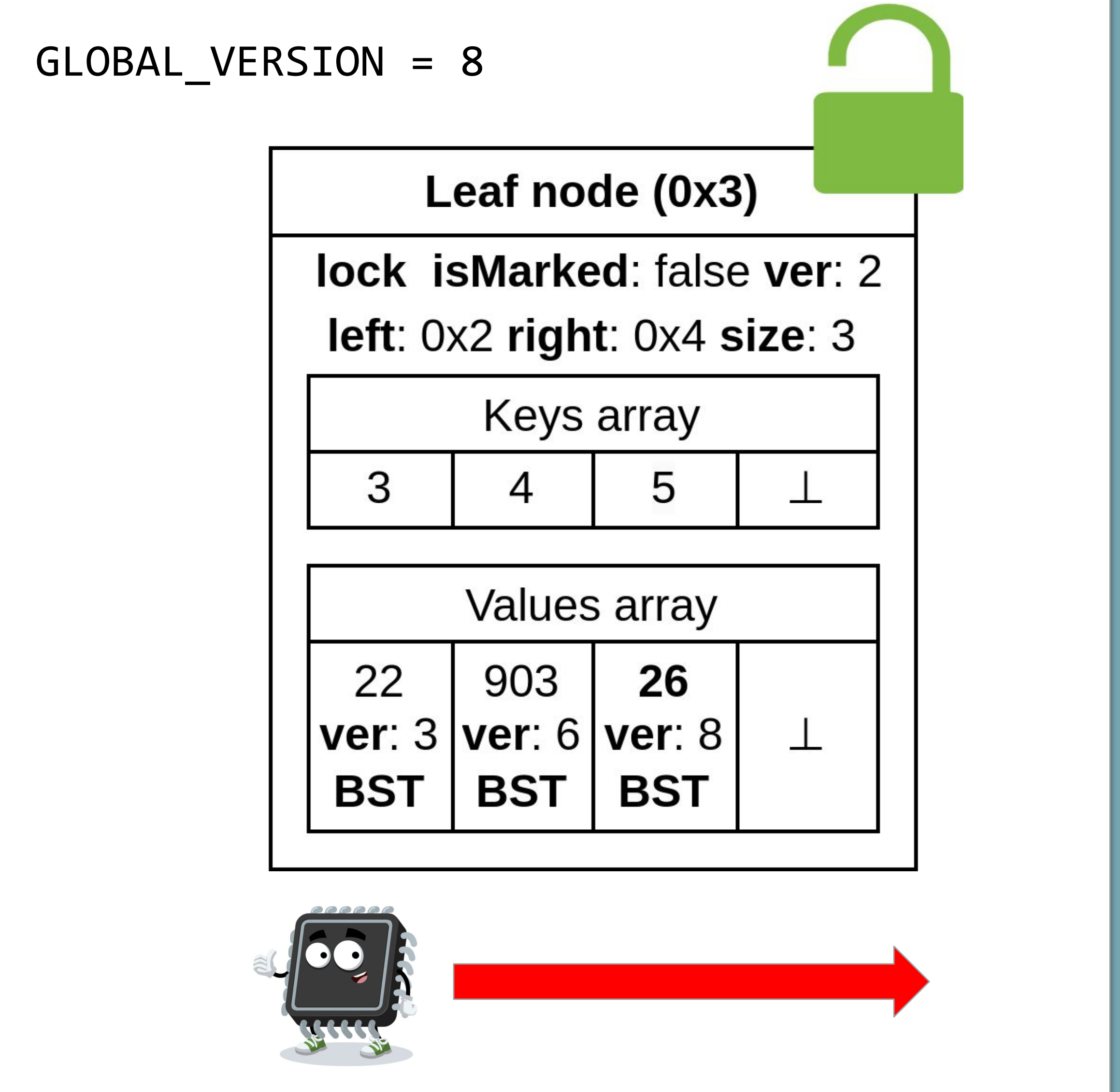


Figure 1. a=2, b=4. A thread scans a leaf node from left to right, gathering values with the most recent version that is less than or equal to 7. In this scenario, it will collect the values 22 and 903.

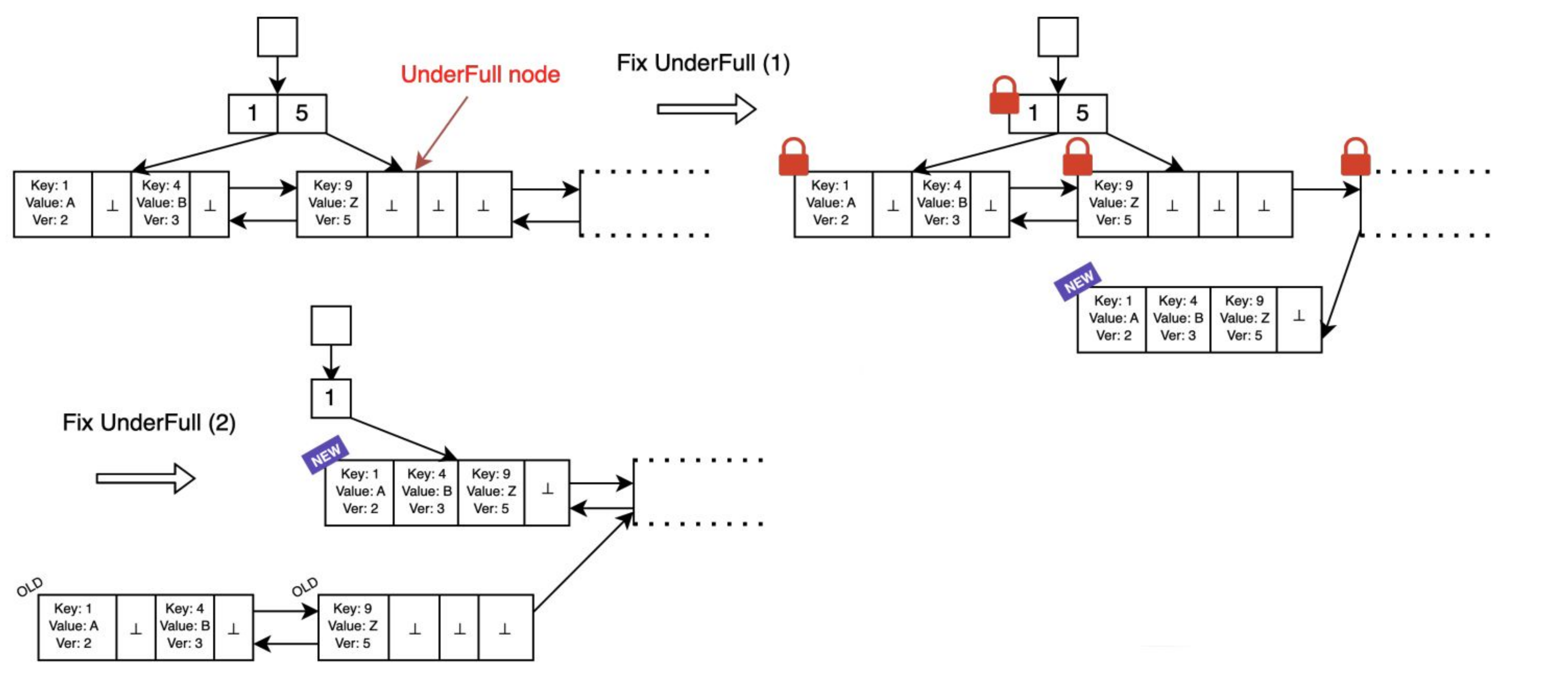
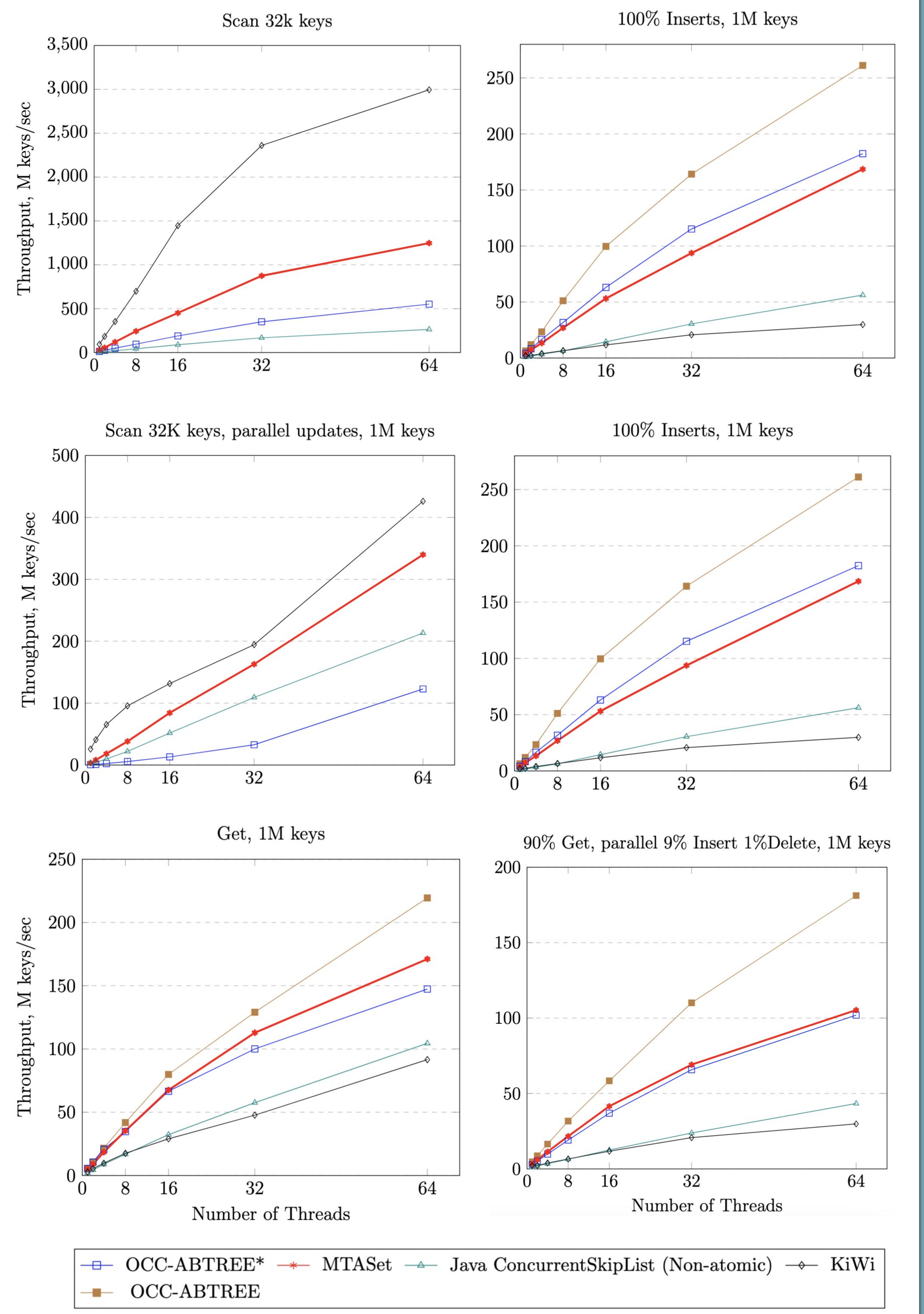


Figure 3. a=2,b=4. An underfull leaf node is being merged with its sibling.

MTASet maintains a **GLOBAL_VERSION** integer variable, which is atomically read and incremented (F&I) by the Scan operation. This version number is used by the scan to determine which values to collect and is read by update operations to assign to the updated values.


Scan 32k keys
100% Inserts, 1M keys
Scan 32K keys, parallel updates, 1M keys
100% Inserts, 1M keys
Get, 1M keys
90% Get, parallel 9% Insert 1%Delete, 1M keys
Throughput, M keys/sec
Number of Threads
OCC-ABTREE*
MTASet
Java ConcurrentSkipList (Non-atomic)
KiWi
OCC-ABTREE