Lock-free Data Structures for Data Stream Processing

ALEXANDER BAUMSTARK

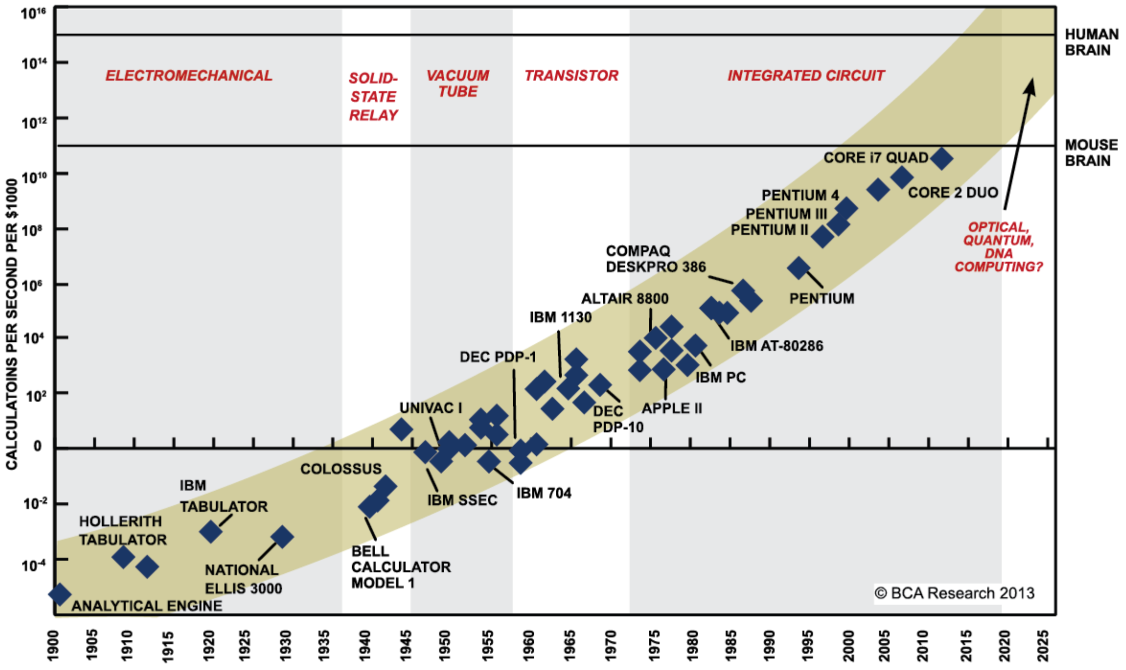
TECHNISCHE UNIVERSITÄT
ILMENAU

18. BTW 2019
Universität Rostock - 600 Jahre

Agenda

1. Motivation
2. Contribution
3. Lock-free Design Principles
4. Data Stream Processing
    1. Tuple Exchange
    2. Symmetric Hash Join

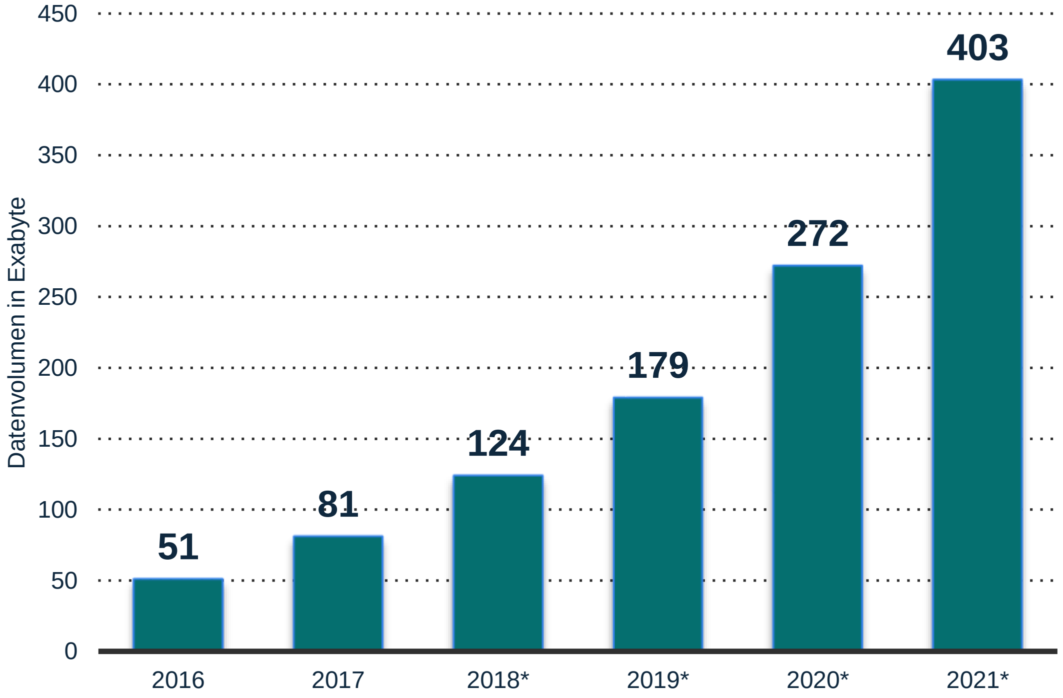
Motivation



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Ray Kurzweil, „The Singularity is near: when humans transcend Biology. „Moore's law“

- https://www.extremetech.com/extreme/210872-extremetech-explains-what-is-moores-law



Big-Data-Datenmenge in Rechenzentren weltweit; 2016 bis 2021 (in Exabyte)

- Cisco Global Cloud Index, 2016-2021

Requirements for Data Processing

- Maximum utilization of parallelism
- Scalable and portable algorithms
- High availability

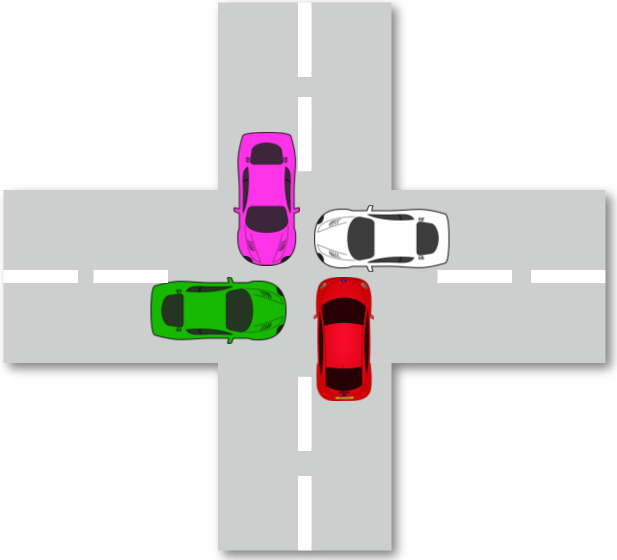
→ **Multithreading**

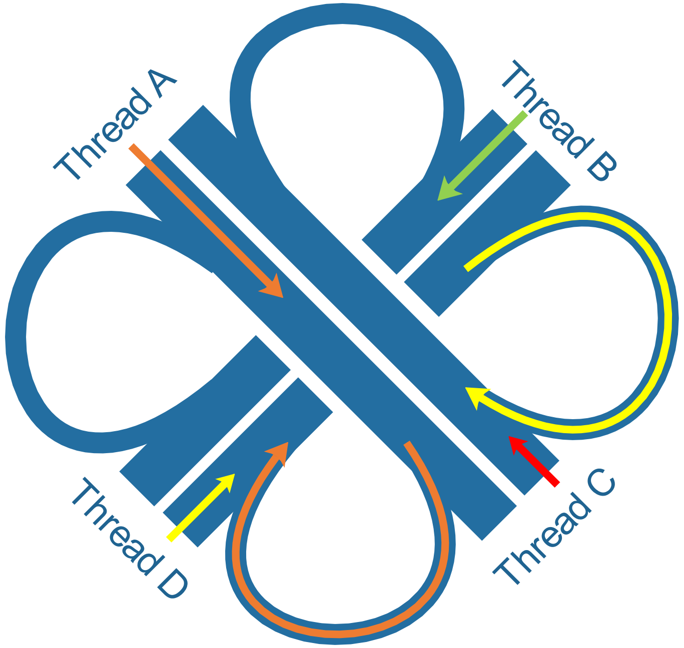
- Instantaneous processing on the fly
- high throughput
- Low latency

→ **Data Stream Processing**

Synchronization

- Goal: Achieve concurrency and consistency
- Two techniques
  - Blocking
  - Non-blocking





„Lock-free Databases“

"From the perspective of the applications written on top of RethinkDB, the **system is essentially lock-free**— you can run an hour-long analytics query on a live system **without blocking** any real-time reads or writes." - *RethinkDB* FAQ

"MemSQL’s storage engine uses multi-version concurrency control with **lock-free skip lists and lock-free hash tables** which allow highly concurrent reads and writes at **very high throughput**." - *MemSQL* FAQ

Contribution

1. Lock-free Tuple Exchange Algorithm

2. Lock-free Multi-Hashmap Design

3. Improved algorithm in Pipefabric with lock-free synchronization
    - Tuple Exchange
    - Symmetric Hash Join

Data Stream Processing

- Requirements: High throughput and low latency
- Achieved with parallelization of tasks and operators

Pipefabric

- Data Stream Processing Engine
- Databases and Information Systems Group/TU Ilmenau
- Supports different protocols: ZeroMQ, MQTT, AMQP
- Join Operators, Window-based Operators
- Concurrent operators and algorithms use blocking synchronization

→reduced degree of parallelism

Design Principles of Lock-free Synchronization

- Key = Atomic Instructions/Read-Modify-Write Operation
- Transaction Properties
  - ACID
  - Linearizability (=Atomic Consistency)
- Atomic Operations:
  - Compare and Swap (CAS)
  - Load-Linked/Store-Conditional (LL/SC)
  - Fetch and Op (FAO), $Op \in \{Add, Sub, Or, Xor\}$

→Guarantee that one out of many contending threads will make progress in a finite number of steps.

Atomic Operation

boolean CAS(value*, value_expected*, new_value)

- Only replace with new value if current value is equal expected value
  - returns false if comparison failed; true otherwise
- Techniques to synchronize
  - Loop until success: refresh current and expected value
- ABA-Problem
  - CAS Operation can’t observe all modification in the interim
  - E.g. A → B → A
  - Solutions: Tagged-Pointers, Hazard-Pointers, Multi-CAS

Classification

Non-blocking: failure or suspension cannot cause failure or suspension of another thread.

Lock-free: guarantees that at least one thread is doing progress on its work.

Wait-free: guarantees that every call finishes its execution in a finite number of steps.

Tuple Exchange

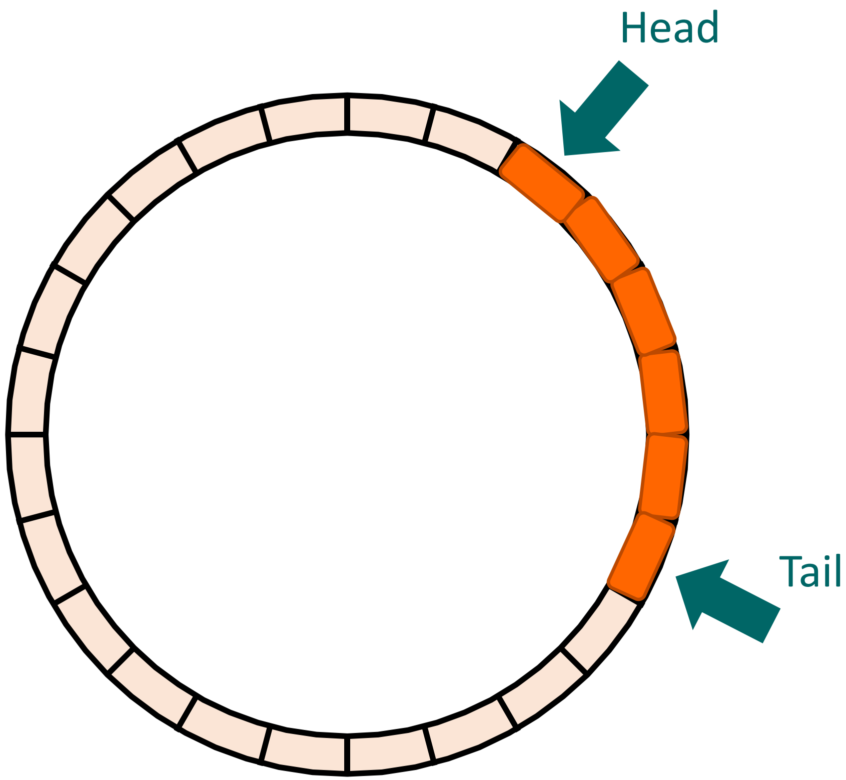
- Threads need to exchange tuples
  - Window Operator
  - Hash Join

- Single-Producer/Single-Consumer
  - Data Structure: Queue
- Pipefabric
  - Uses STL Queue
  - Locks with condition variables

Tuple Exchange

Lock-free SPSC Queue → Ringbuffer (fixed sized array)

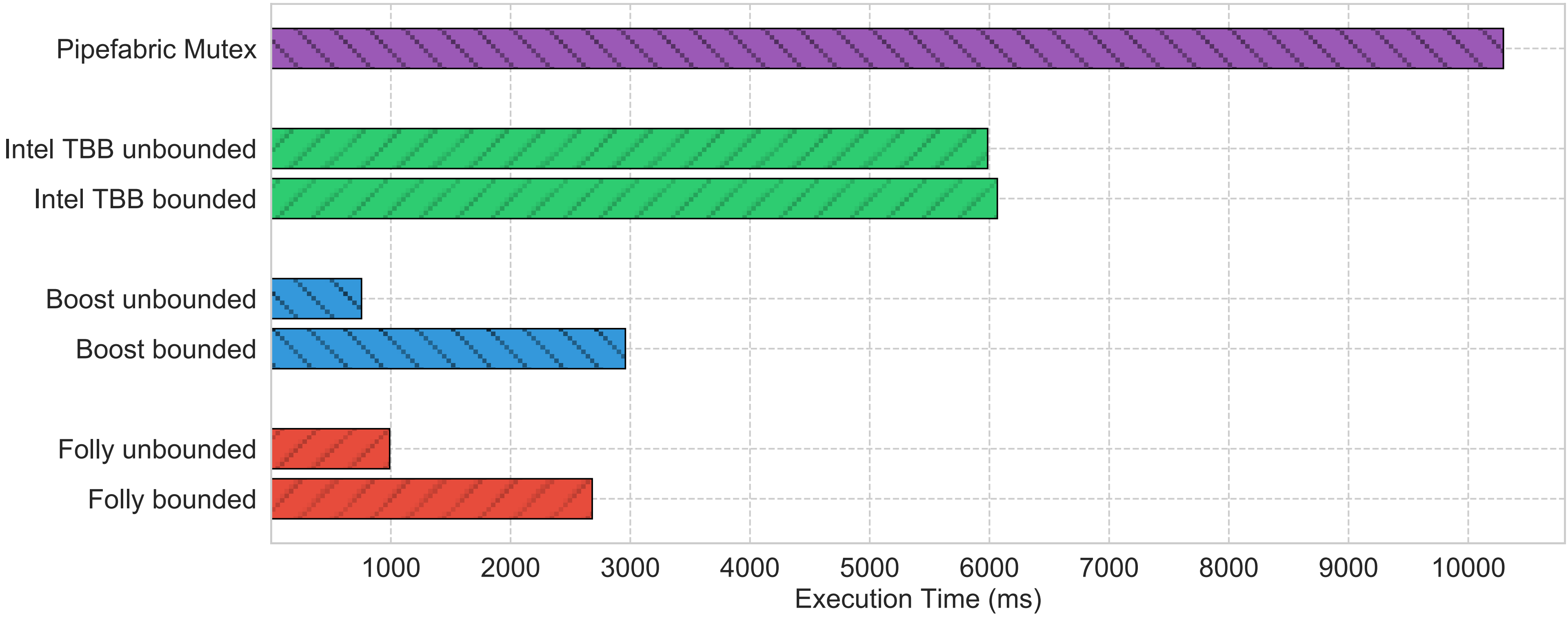
- Push & Pop on different locations
  - Atomic load/store operations sufficient
  - + memory order
- Consistency check before each store
- Better results than node-based (true unbounded) implementations



Tuple Exchange Benchmark

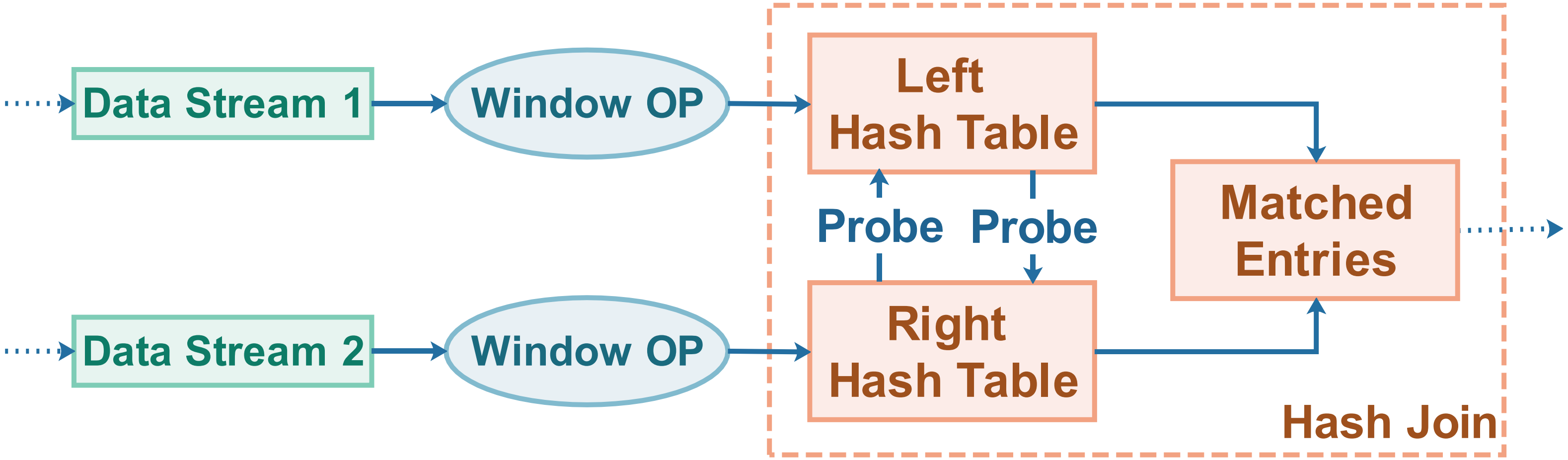
- Benchmark with 3 non-blocking queue implementations
  - **Boost** SPSC Queue
  - **Intel Threading Building Blocks** Reader-Writer Queue
  - **Facebook Folly** Producer-Consumer Queue
- 2 Benchmarks: Bounded (1024) + „Unbounded“ (Maximum size)
- Producer: 10 Million push Operations
- Consumer: pop until 10 Million tuples
- System: Intel Xeon Phi (Knights Landing) 7210
    - 64 Cores / 256 Threads @ 1.3-1.5GHz

Tuple Exchange Benchmark



Symmetric Hash Join

1. Input: two tuple streams
2. Each tuple (key) is hashed to the appropriate hashmap
3. Hashmaps probe each entry with other
4. Matched entries continued as resulting stream



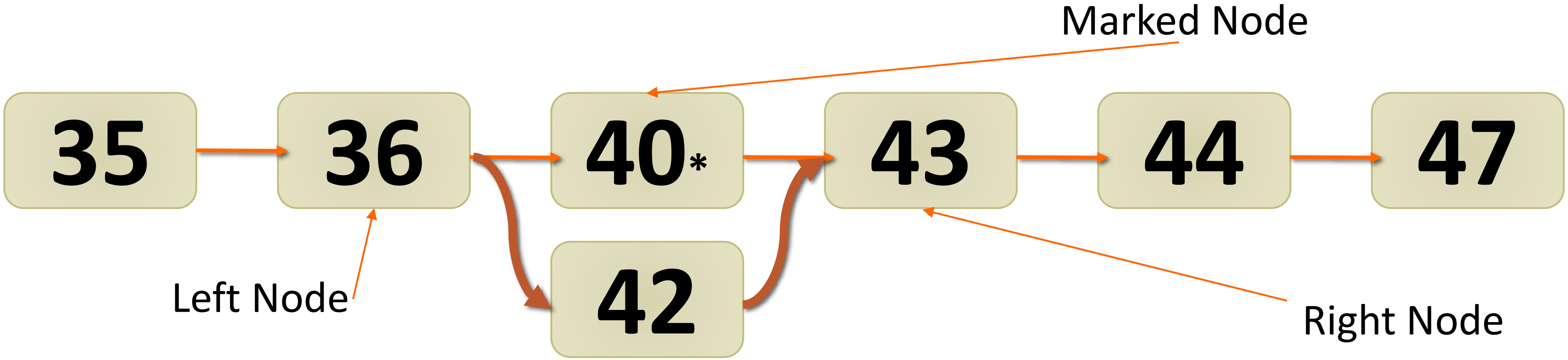
Symmetric Hash Join

- Base: Multi-Hashmap
- Support same key elements
- Implementation in Pipefabric:
  - based STL unordered multimap
  - Threads locks the entire map to operate
  - no real parallelism with higher thread numbers

→Lock-free Multi-Hashmap

Lock-free Multi-Hashmap Design

- Basis: Lock-free Linked List



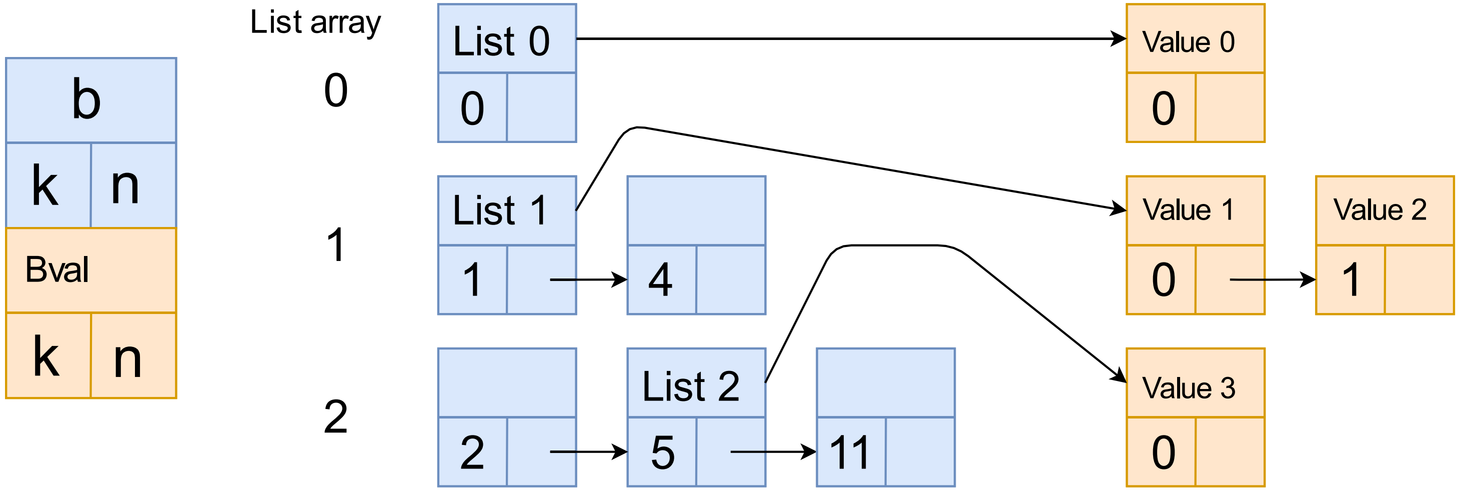
Main Method: Search Procedure

- Searches for the position of the left and right node to delete or insert node between then
- Deletes logically deleted marked nodes
  - no ABA-problem

Harris, Timothy L. "A pragmatic implementation of non-blocking linked-lists." *International Symposium on Distributed Computing*. Springer, Berlin, Heidelberg, 2001.

Lock-free Multi-Hashmap Design

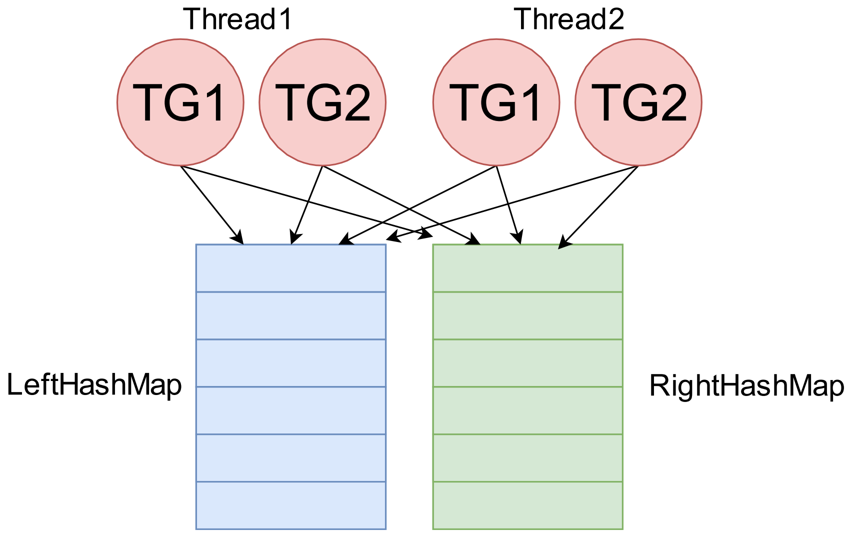
- Multi-Hashmap
  - Each hash points to a linked list
  - Each node points to a further list (Bucket)



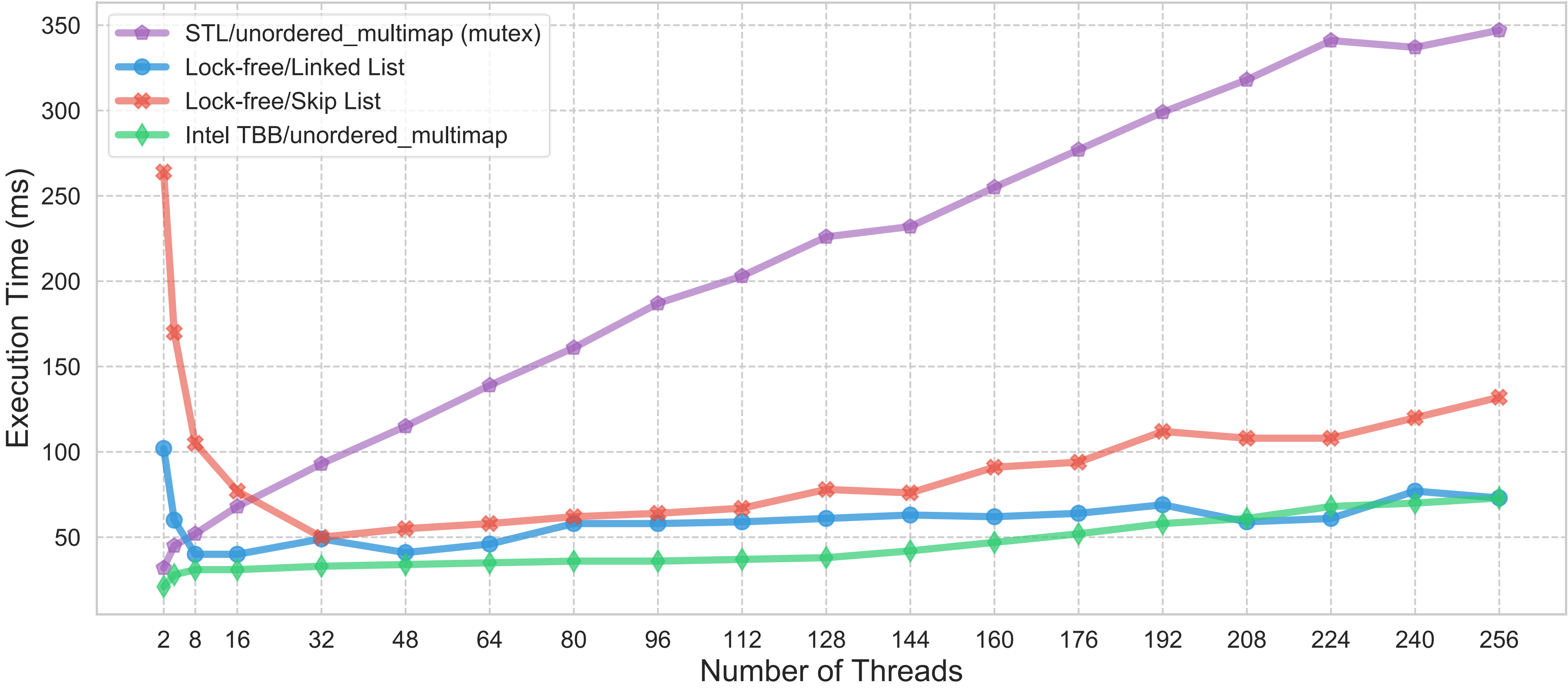
- Array structure needs initialization!

Symmetric Hash Join Benchmark

- 2 lock-free implementations
  - Linked Linked based
  - Skip List based
- Additional blocking (fine-grained) implementation
  - based on unordered_multimap structure from Intel TBB
  - Equivalent to STL unordered_multimap
- Benchmark:
  - 2-256 threads
  - equal distributed 10.000 tuples



Symmetric Hash Join Benchmark



Conclusion

- Scalable and robust algorithms
- Guarantee for progress
- Lock-free data structures fulfill the requirements
  - High throughput
  - Low latency
- Overhead for additional structures (e.g., Marked Pointer)
- Need non-blocking memory management
- Fine-grained locking can achieve same performance results