

# Hardware + Algorithms = Seriously Concurrent Hash Tables

Xiaozhou Li (Princeton), David G. Andersen (CMU), Michael Kaminsky (Intel Labs), Michael J. Freedman (Princeton)

## Goal: Fast Concurrent Read & Write

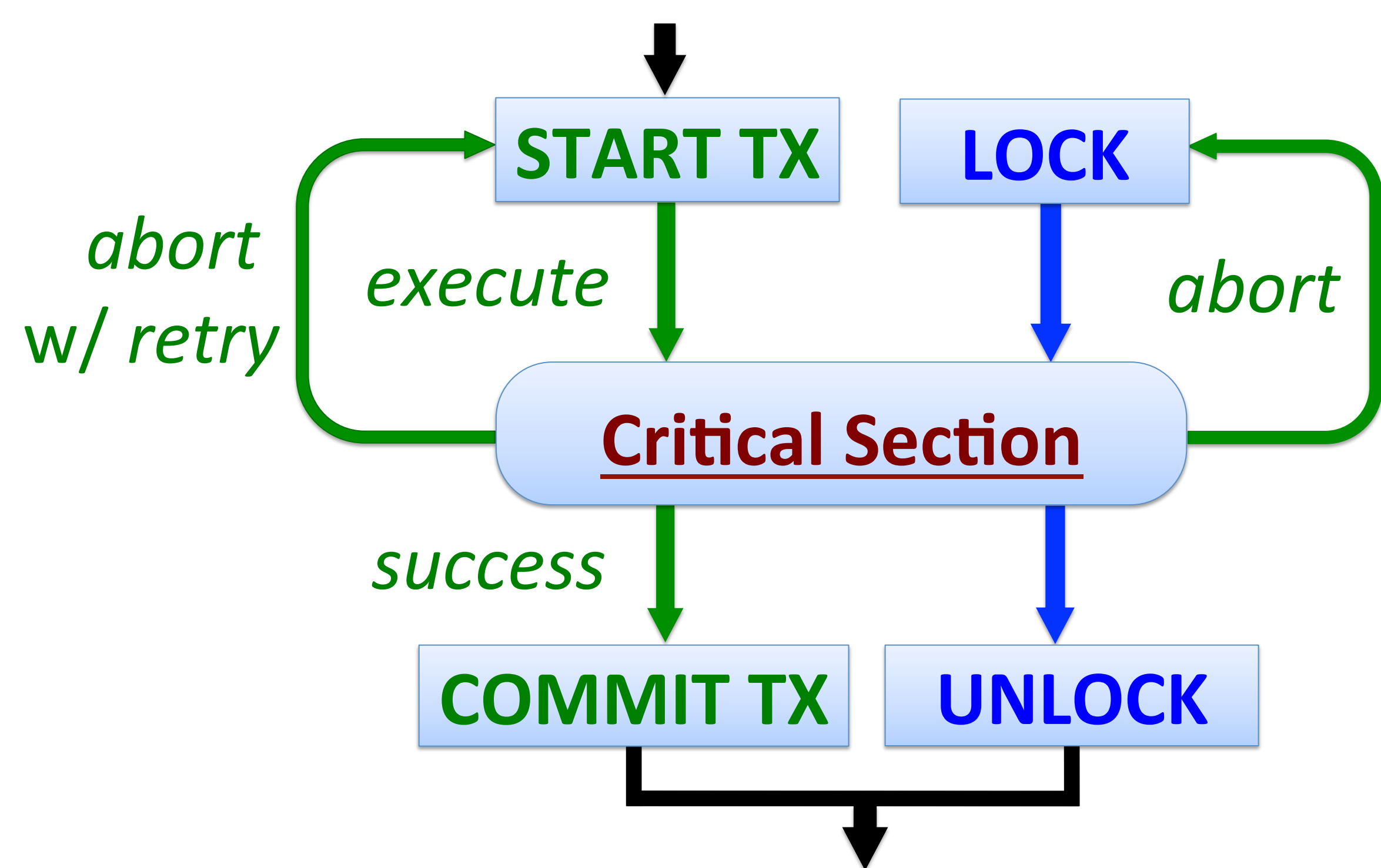
- A. Memory efficient (e.g., 95% space utilized)
- B. Fast concurrent reads
- C. **Fast concurrent writes** (scale w/ # of cores)

## Technique Overview

- Support concurrent writers with **Intel TSX** (*hardware transactional memory*)
- Minimize critical sections
- Exploit data locality

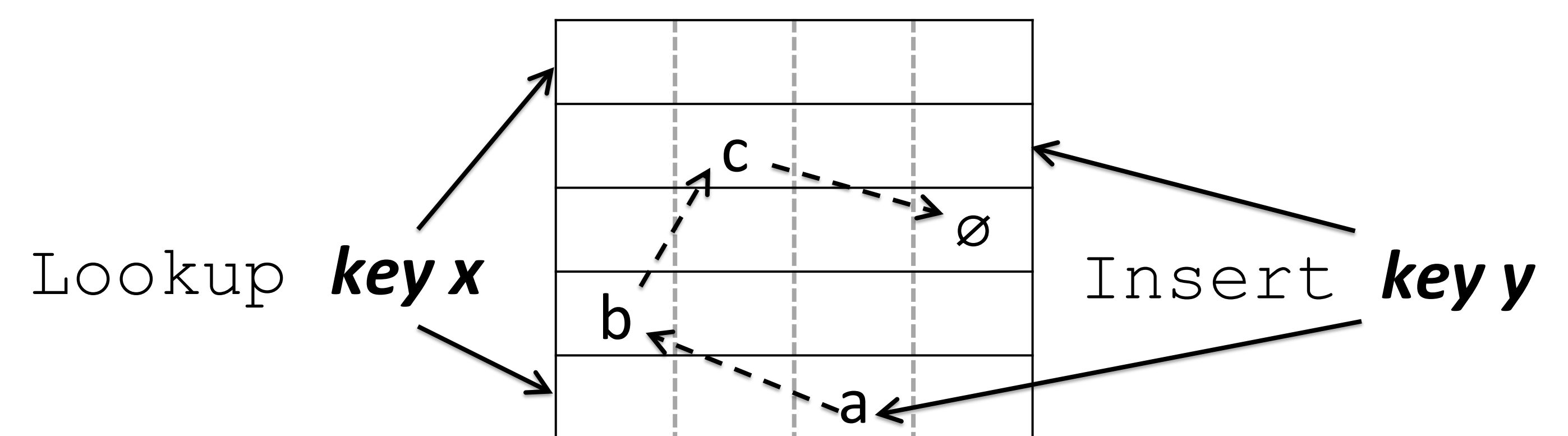
## Implement Intel TSX Lock Elision

Optimized for short transactions (TX)



## Starting Point: Optimistic Cuckoo Hashing

MemC3 (NSDI '13): goals A and B, but NOT C



Each key is mapped to two random buckets

Lookup ( $x$ ):

Read 2 buckets and compare w/ each slot.  
Use key version counters, no locking on read path.

Insert ( $y$ ):

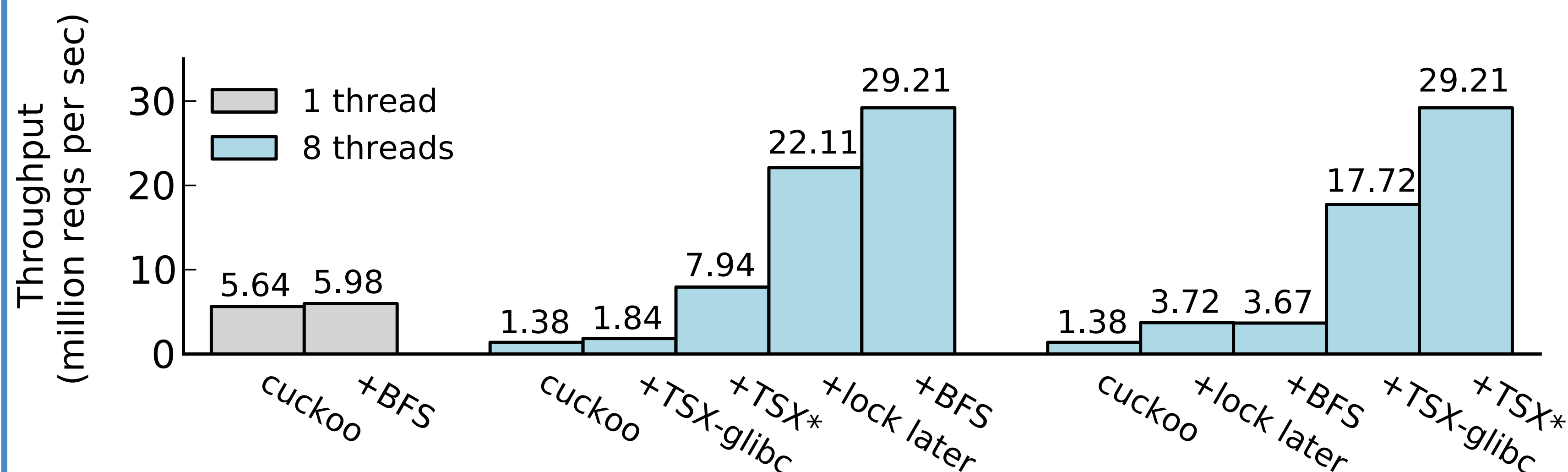
May need to **displace** existing keys (e.g.,  $a, b, c$ ) to alternate bucket. **Lock the hash table** – single writer

## Algorithmic Optimizations (for Insert)

- **Lock after discovering an empty slot**
  - minimize critical sections
- **Breadth-first search for an empty slot**
  - fewer items displaced, enables prefetching
- **Increase set-associativity**
  - less random (more sequential) memory reads

## Preliminary Results (2 GB hash table, ~134.2 M entries, 8 Byte keys and 8 Byte values)

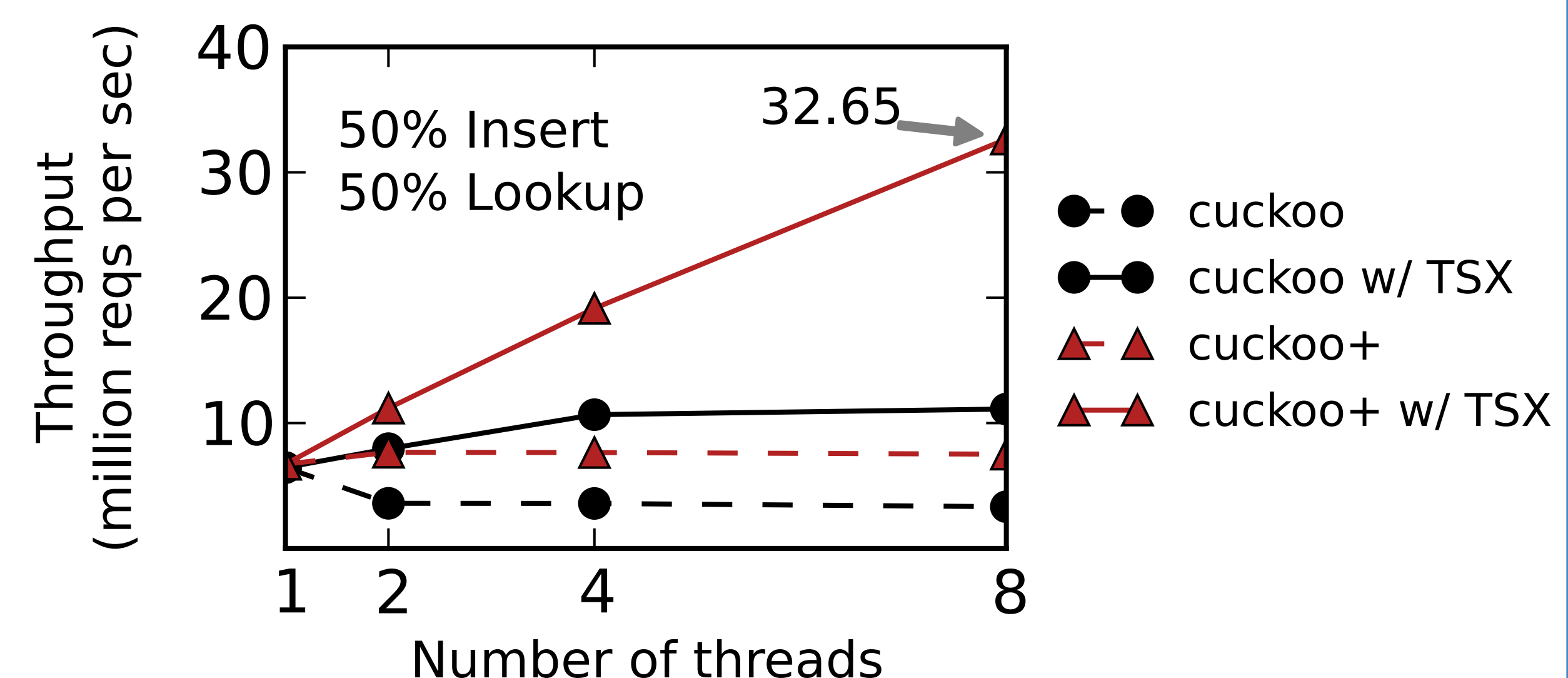
Platform: Intel Haswell i7-4770 @ 3.4GHz, 4 cores (8 hyper-threaded cores), 16 GB DRAM, 8 MB L3-cache



Performance under 100% Insert (load 0-0.95)

TSX-glibc: Intel library

TSX\*: our optimized impl.



Throughput vs. # of threads (load 0-0.95)

