Concurrent Skiplists

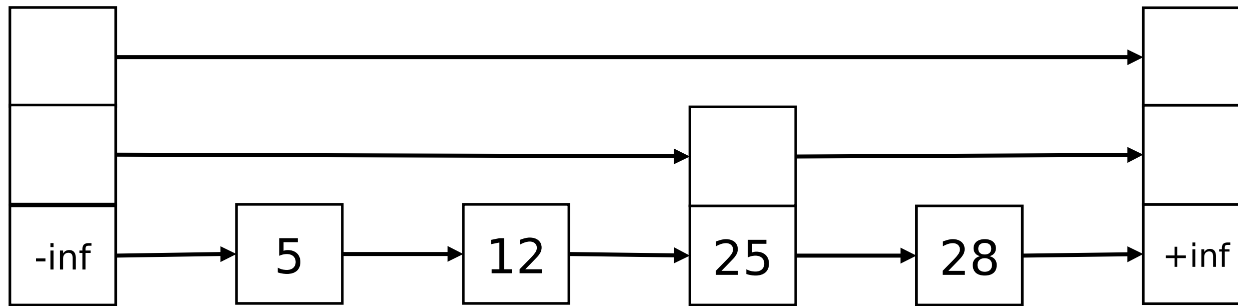
Final Presentation

Christine Zeller
Karolos Antoniadis

Goal

- Pick a concurrent data structure: Skiplist
  - Apply the techniques from the lectures
  - Get a better understanding of concurrency
  - Write a parallel implementation that scales well

Recap - Skiplists



insert

find

delete

O(log n) in Expectation

Related Work

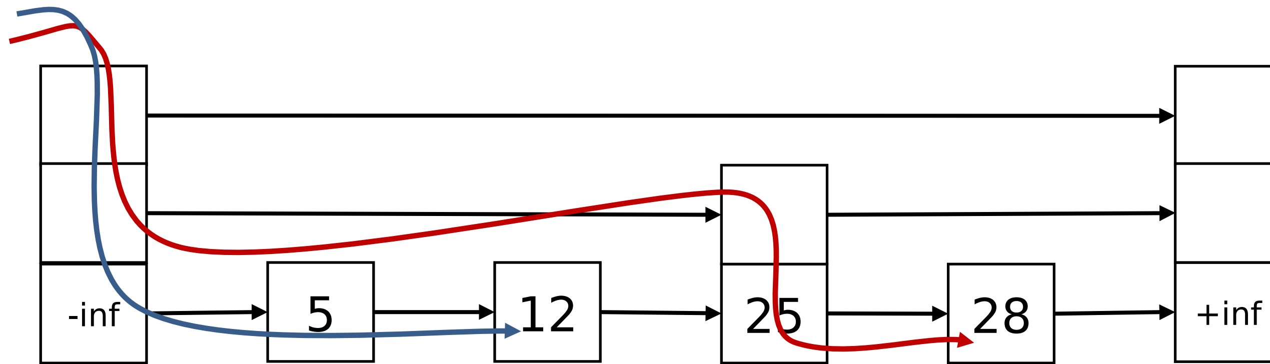
- A Simple Optimistic Skiplist Algorithm by Y.Lev et al.
  - Fine-grained locking

- A lock-free concurrent skiplist with wait-free contains operator by M. Herlihy et al.

Outlook: What we did

- Solved the deadlock!
- Lock-free implementation
- Lock-based with backoff
- Experiments

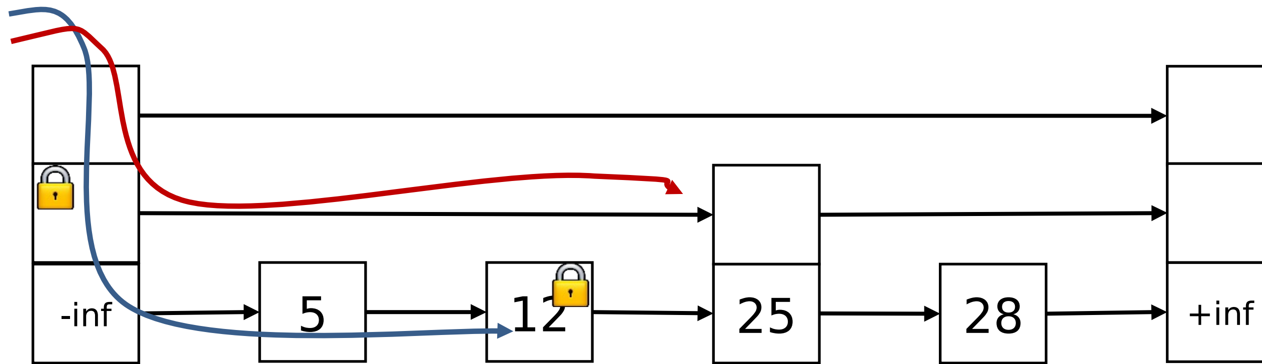
Recap - Lock-based



find 12

find 29

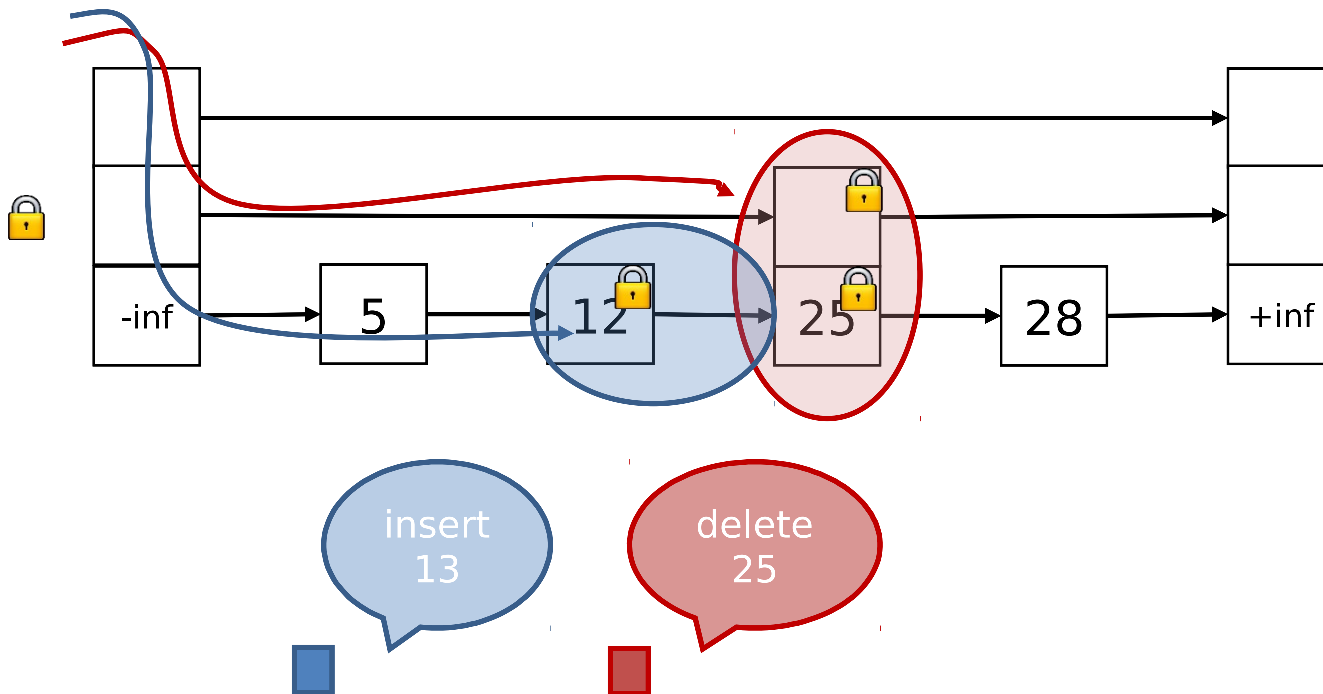
Recap - Lock-based



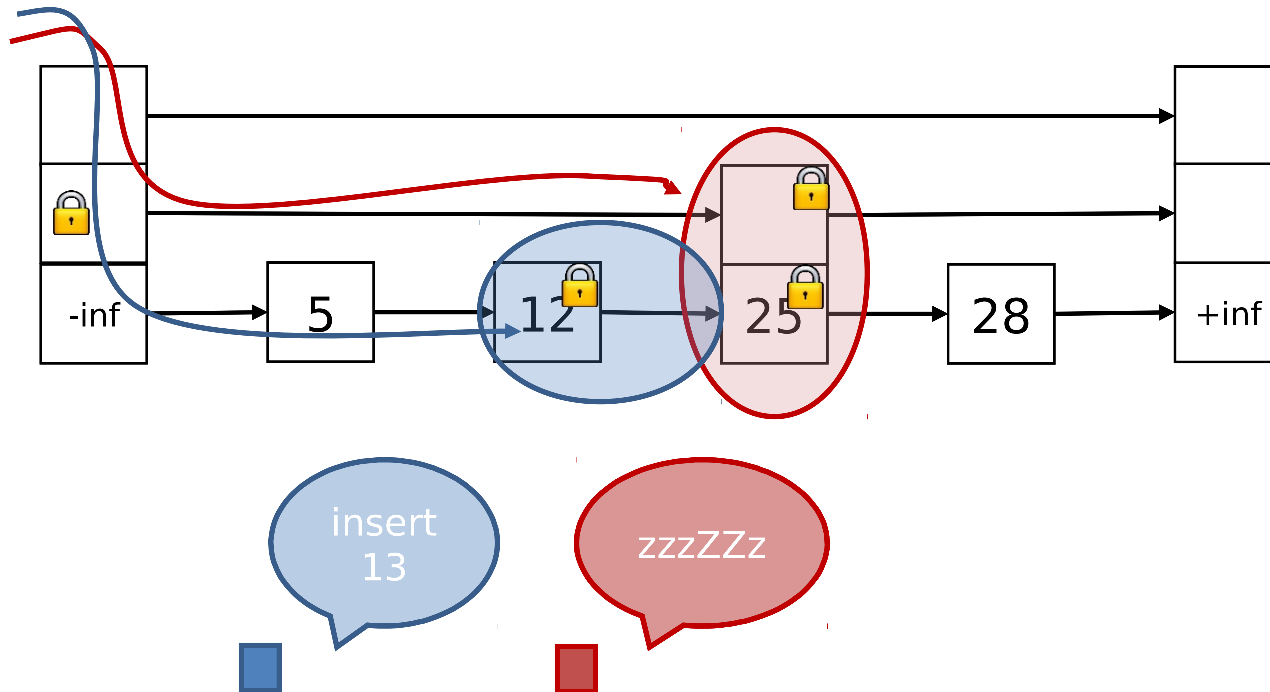
Insert
13

find
25

Recap - Lock-based



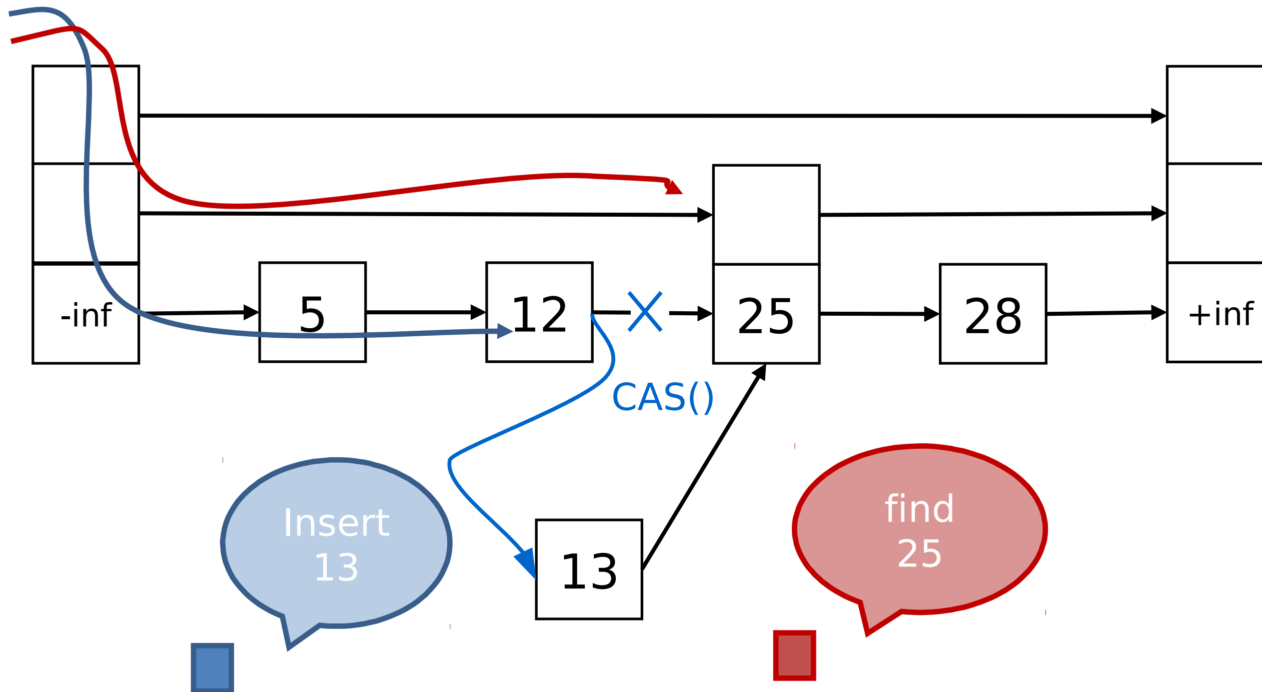
Lock-based with backoff



Lock-based with backoff

- Use exponential backoff
  - Try to lock the node
  - Sleep if unsuccessful
  - Increase sleep time if repeatedly unsuccessful

Lock-free



Lock-free

- Each level can be seen as a lock-free linked list

- Differences to lock-based skiplist
  - Skiplist property doesn’t (necessarily) hold during execution
  - Helper method removes marked nodes

Testing

- Verify:
  - Skiplist has to be still sorted
  - Skiplist does not contain duplicates
  - Skiplist property holds
  - #of elements = #of inserts - #of deletes

Testing - Linearizability

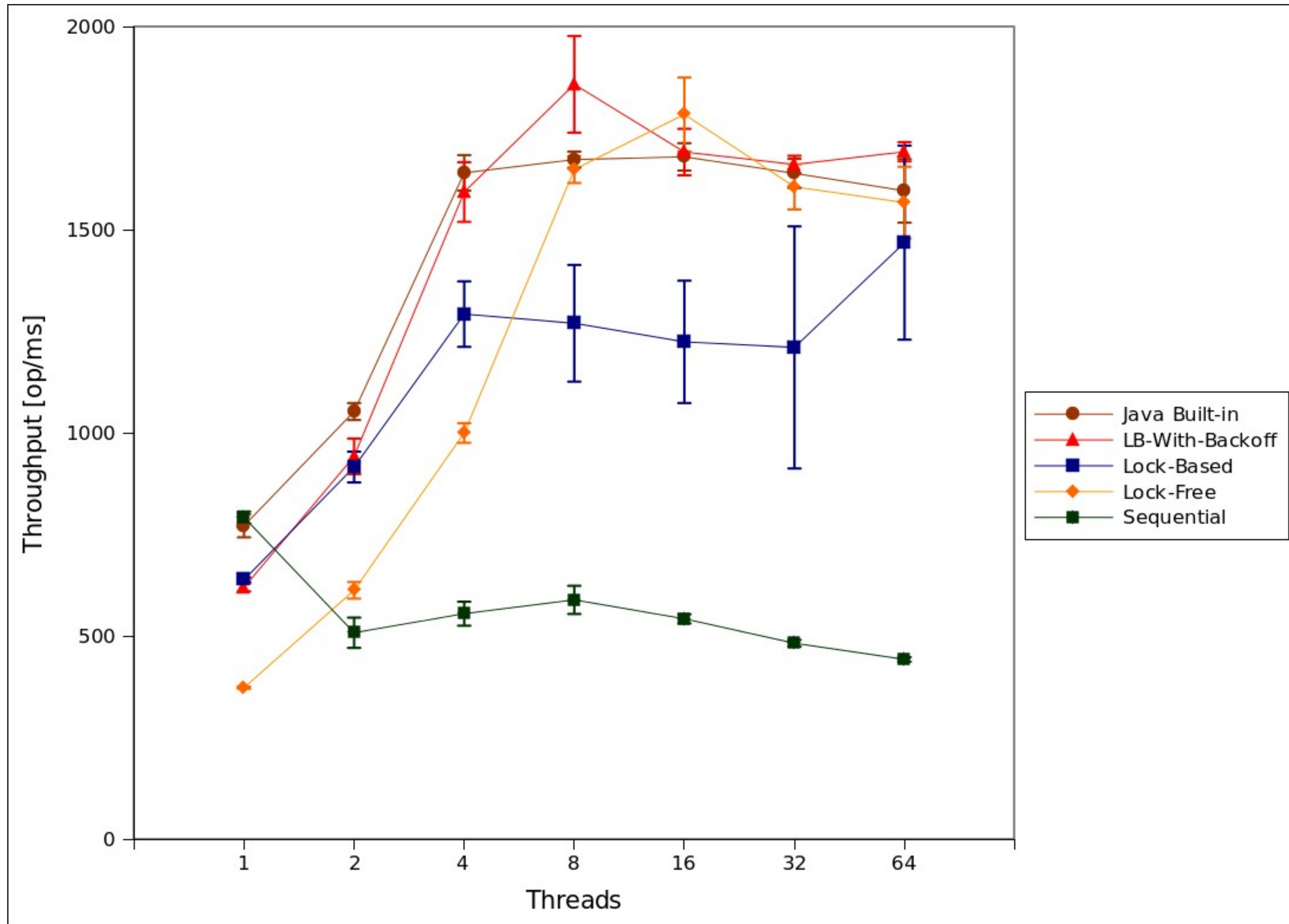
- Lock-based setting only
- Linearization points (l.p.) are explicit
  - Every thread logs its l.p.
  - Merge all logs and sort them by time
  - Execute the logs sequentially

- Placing l.p. for unsuccessful operations hard

Experimental Setup

- Kanifushi: 32-cores
- Benchmarks:
  - Fixed number of operations per thread
  - Fixed range of numbers
  - Fixed percentage of insertions, deletions and finds
  - Compare lock-based, lock-based with backoff, lock-free and sequential to ConcurrentSkipListSet (Java)

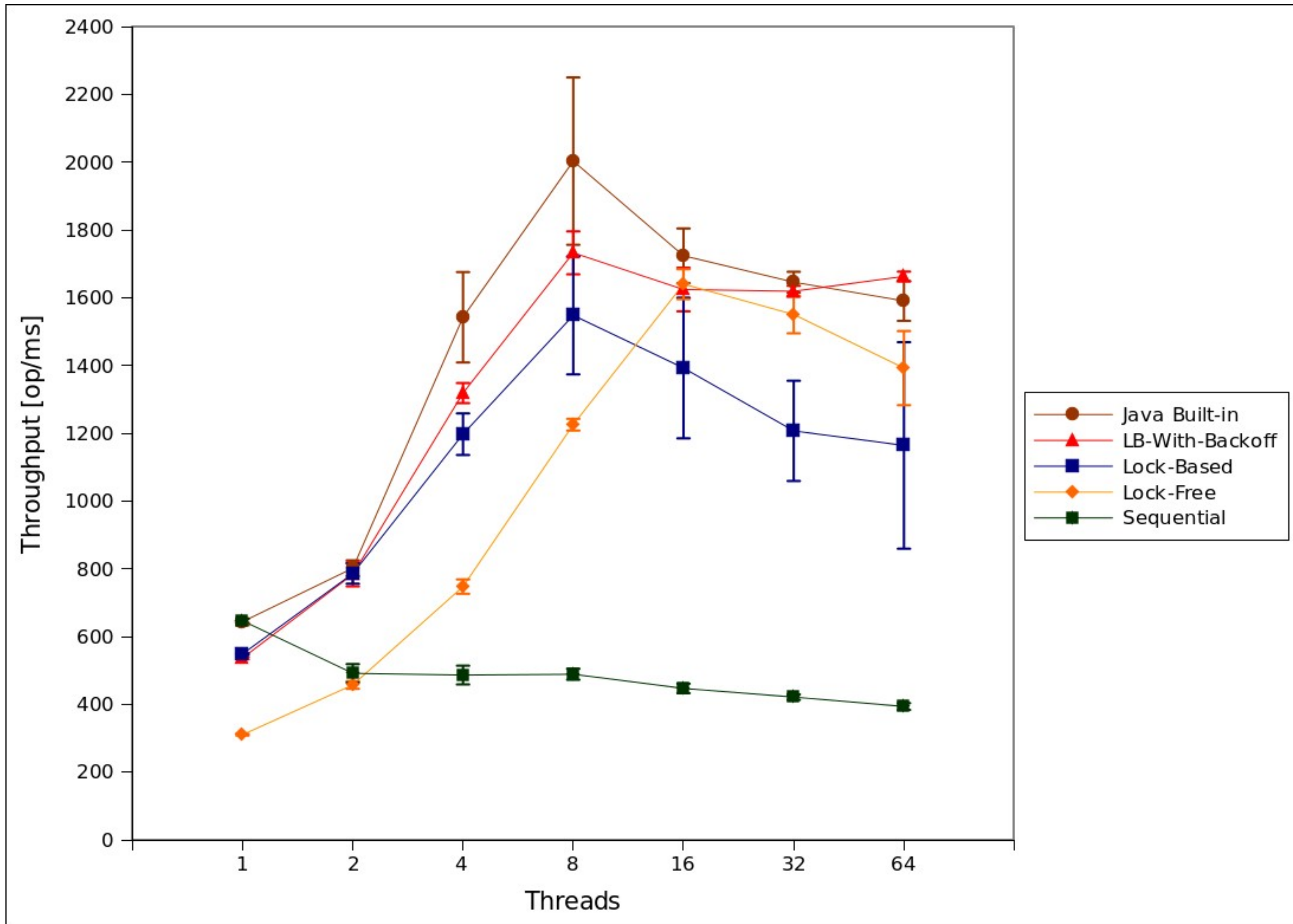
Results



9% insertions, 1% deletions, 90% finds

Range: 2’000’000 numbers, 1’000’000 operations

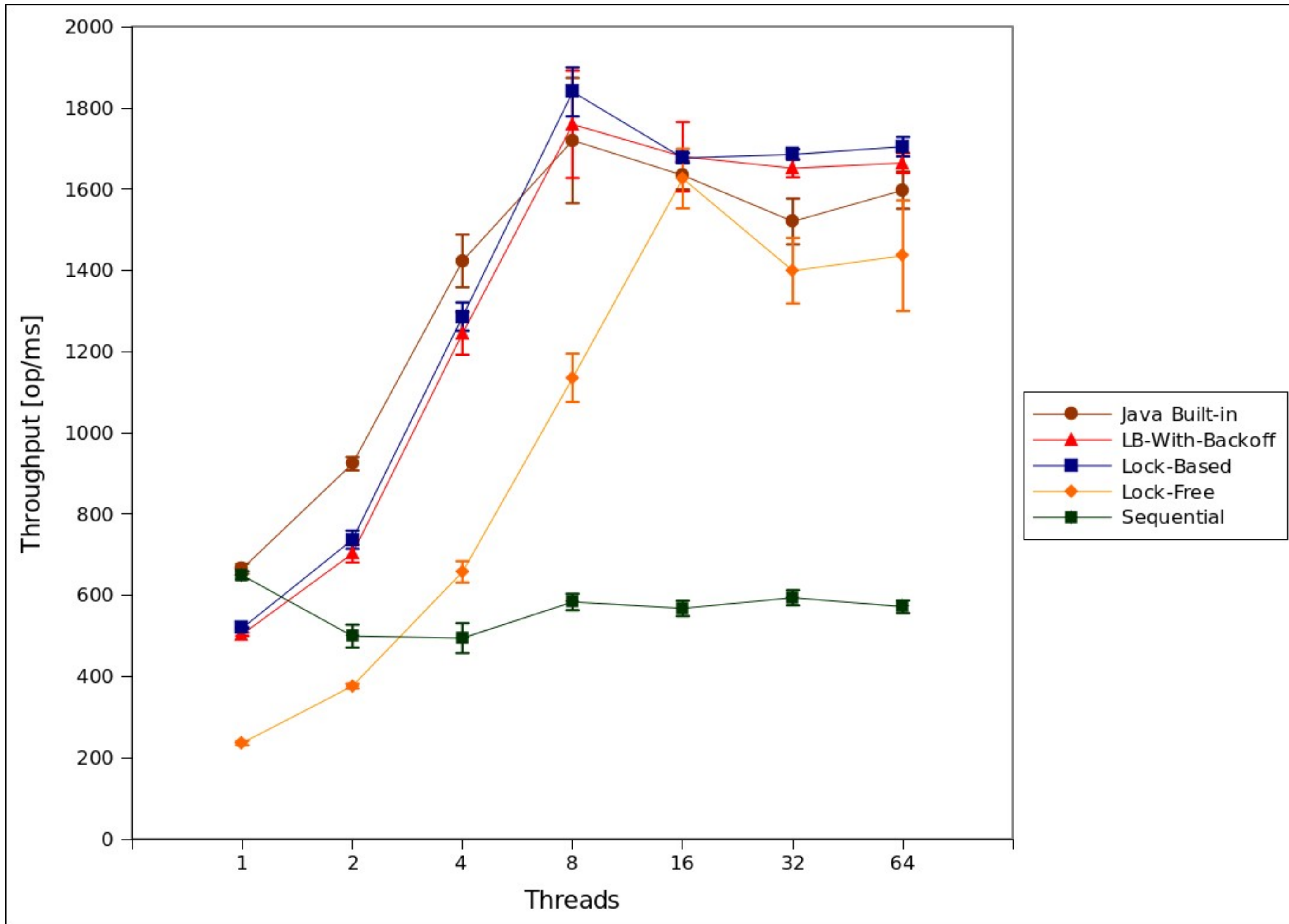
Results



20% insertions, 10% deletions, 70% finds

Range: 2’000’000 numbers, 1’000’000 operations

Results



50% insertions, 50% deletions, 0% finds

Range: 200’000 numbers, 1’000’000 operations

Conclusion

- Backoff did not help as much as we thought
- Compared to Java built-in, we’re doing good
- Input is artificial

Conclusion

- What did we learn?
  - Parallel programming is hard
    - Deadlocks are introduced fast
  - Better understanding of Java Concurrency, Linearizability, ...

Questions?