# A Specialized B-tree for Concurrent Datalog Evaluation

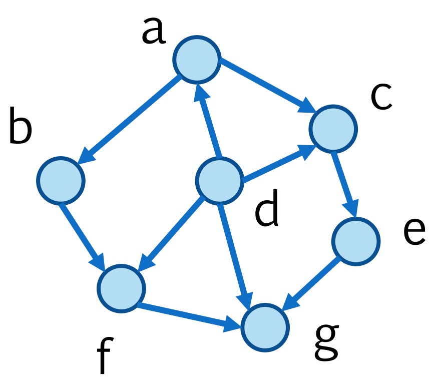
**Herbert Jordan**[1], Pavle Subotić[3], David Zhao[2], and Bernhard Scholz[2]

PPoPP 2019, 16-20 February 2019, Washington, DC

1) University of Innsbruck
2) University of Sydney
3) University College London

DPS
Distributed and Parallel Systems

# Datalog (by Example)



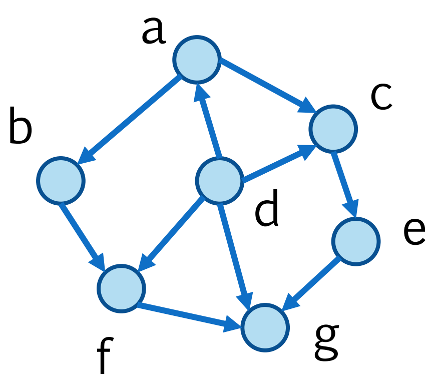

graph

| from | to |
|---|---|
| a | b |
| a | c |
| b | f |
| c | e |
| d | a |
| d | c |
| … | … |

edge relation

Which nodes are connected?

# Datalog (by Example)




graph

| from | to |
|---|---|
| a | b |
| a | c |
| b | f |
| c | e |
| d | a |
| d | c |
| … | … |

edge relation

```
path(X,Y) :- edge(X,Y).

path(X,Z) :- path(X,Y),
             edge(Y,Z).
```

**Datalog**
query

# Datalog

- Benefits:
  - a concise formalism for powerful data analysis
  - lately major performance improvements and tool support

- Applications:
  - data base queries
  - program analysis
  - security vulnerability analysis
  - network analysis

100s of relations and rules,
billions of tuples,
all in-memory

# Query Processing

relations → set of integer tuples

rules → sequence of relational algebra operations on sets

# Example

```
path(X,Z) :- path(X,Y), edge(Y,Z).
```

$$
\begin{array}{l}
delta \leftarrow path \\
\text{while} \; (\; delta \neq \emptyset \;) \, \{ \\
\qquad \left.\begin{array}{l} new \leftarrow \pi(delta \bowtie edge) \setminus path \end{array}\right\} \\
\qquad path \leftarrow path \cup new \\
\qquad delta \leftarrow new \\
\}
\end{array}
$$

computational expensive and dominating part

# Example

$$new \leftarrow \pi(delta \bowtie edge) \setminus path$$

```
Relation new;
for t1 ∈ delta {
   auto l = edge.lower_bound( { t1[1], 0 } );
   auto u = edge.upper_bound( { t1[1]+1, 0 } );
   for t2 ∈ [l,u] {
      Tuple t3 = { t1[0], t2[1] };
       if ( t3 ∉ path ) {
            new.insert(t3);
        }
     }
   }
```

# Example

$$new \leftarrow \pi(delta \bowtie edge) \setminus path$$

```
Relation new;
#pragma omp parallel for
for t1 ∈ delta {
  auto l = edge.lower_bound( { t1[1], 0 } );
  auto u = edge.upper_bound( { t1[1]+1, 0 } );
  for t2 ∈ [l,u] {
    Tuple t3 = { t1[0], t2[1] };
     if ( t3 ∉ path ) {
        new.insert(t3);
     }
    }
  }
```

all read accesses (right hand side)

one write access (assignment)

But: write target is never read on right hand side!

# Needed

- efficient   data structure for relations
  - maintain set of n-dimensional tuples

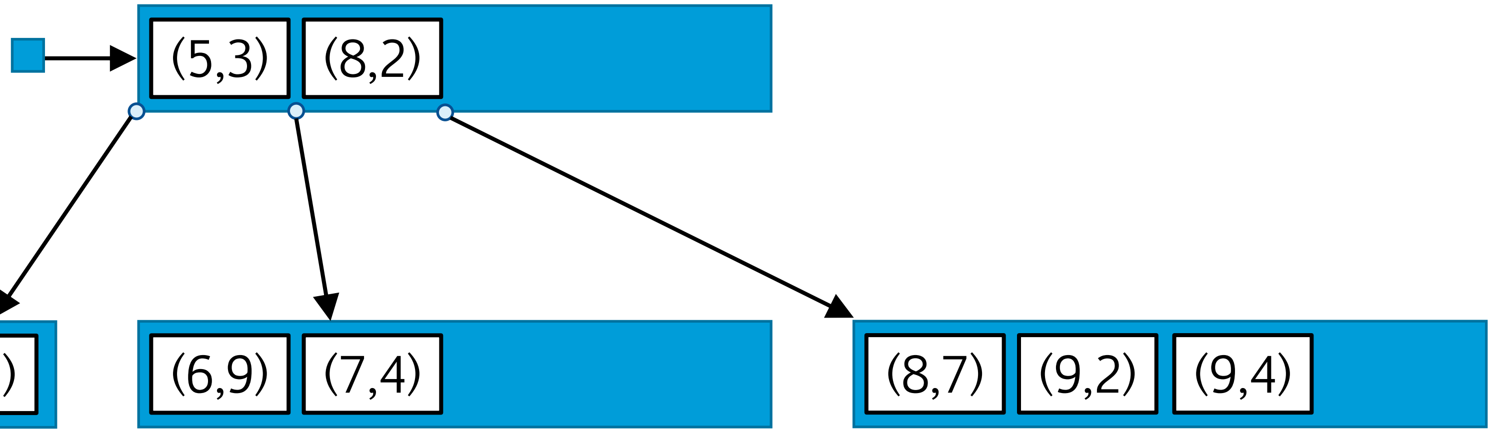
  - efficient support for
    - insertion,
    - scans,
    - range queries,
    - membership tests,
    - emptiness checks

well supported
by **B-trees**

  - efficient synchronization of
    concurrent inserts

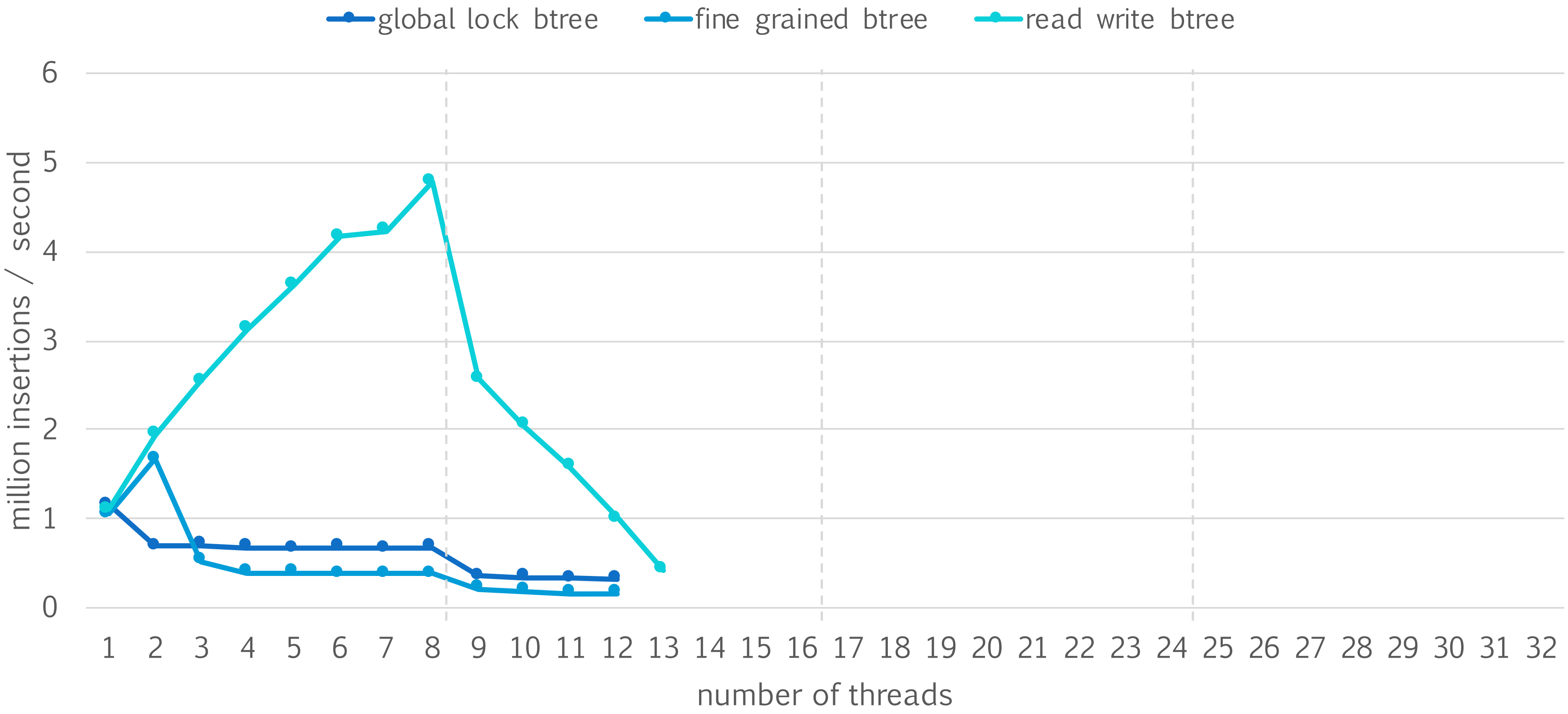
not so much ...

# B-tree



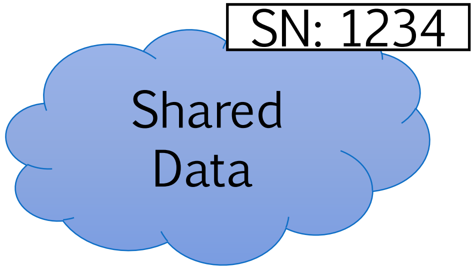

- Insertion:
  - locate target leaf node
  - split leaf node if necessary, may propagate up
  - insert element in sorted leaf-node element array

# B-tree Locking Strategies



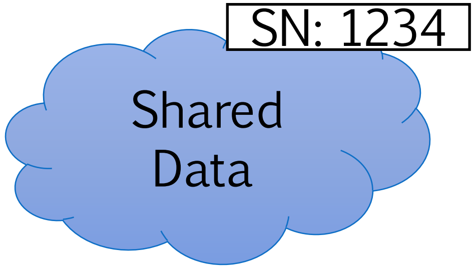

random order, on 4x8 core Intel Xeon E5-4650

# Seqlocks / Optimistic Locking



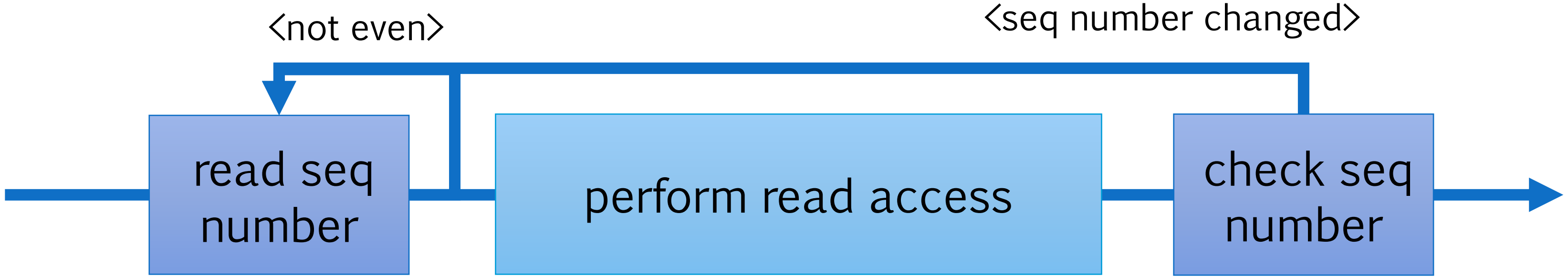

<u>sequence number:</u>

- even: no writer active
- odd: one writer active

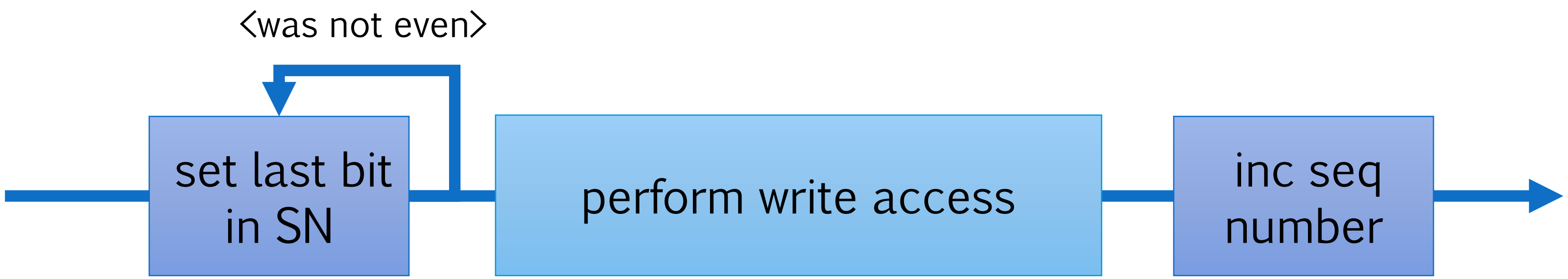


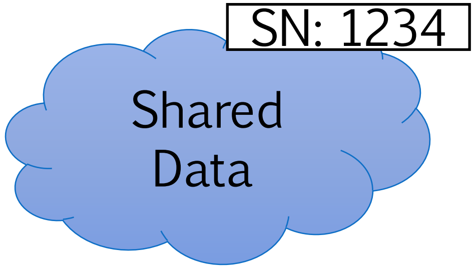

# Seqlocks

› Reader:



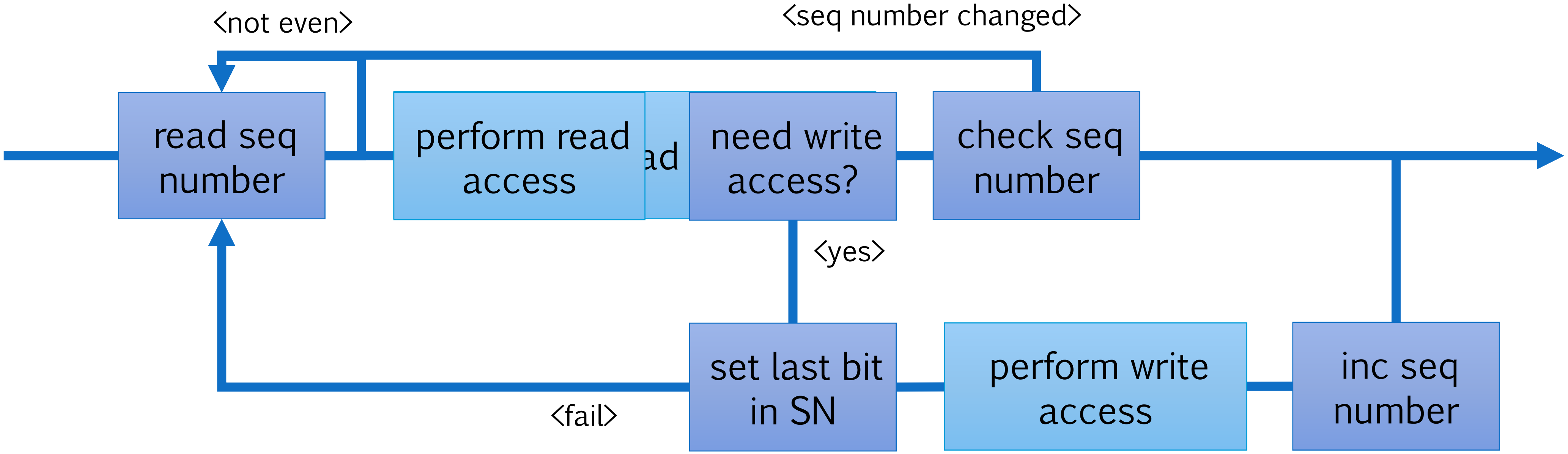

› Writer:






# Optimistic Read/Write Lock

› Reader

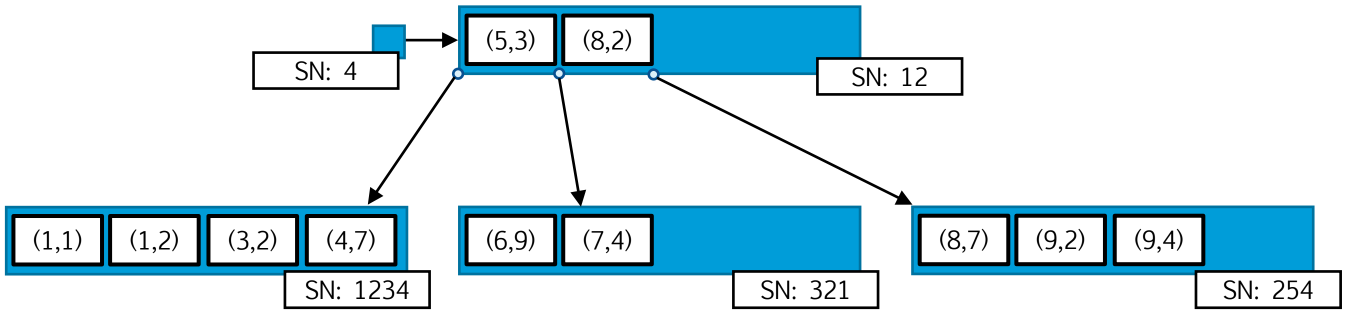


# Optimistic B-tree

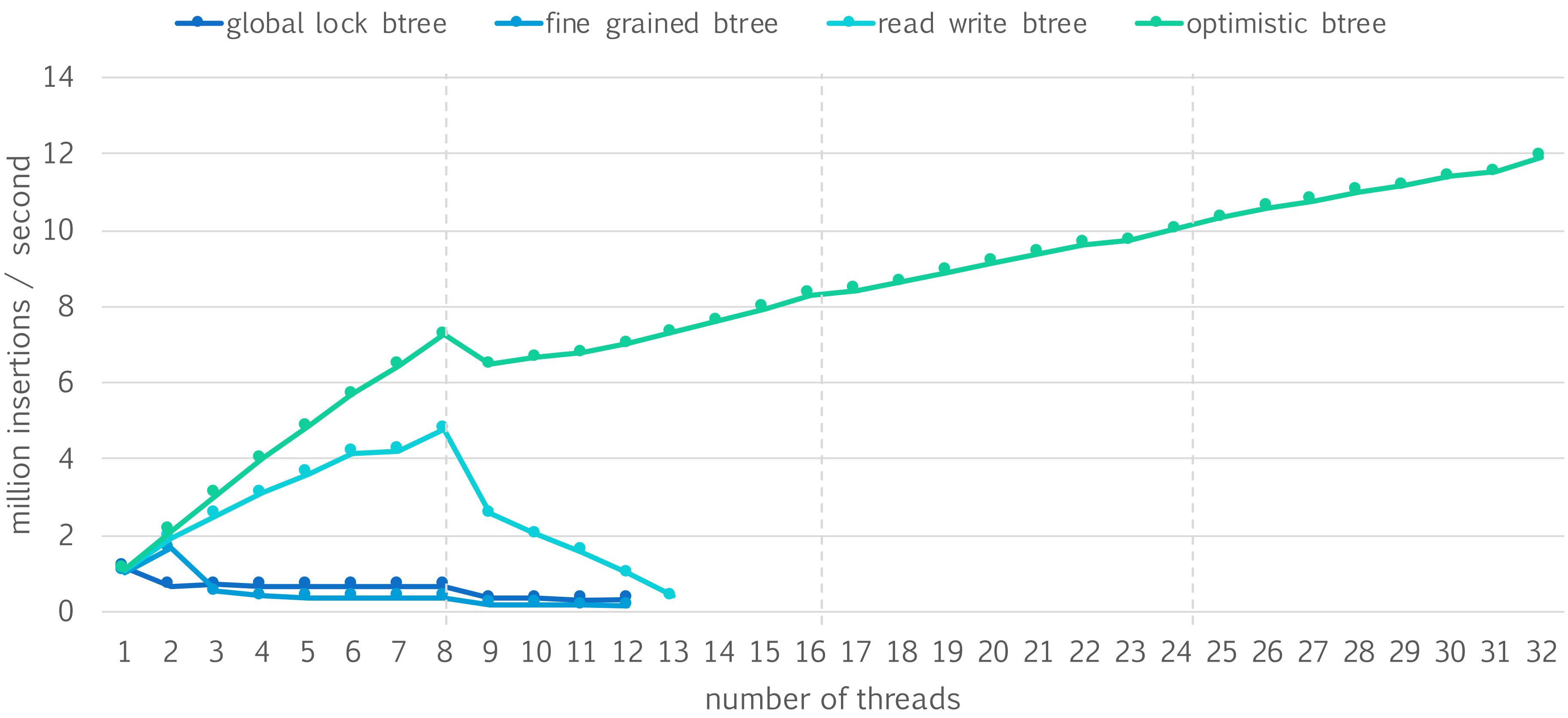
› Protect nodes and root pointer with optimistic R/W lock

› Synchronize insert operation
  – read access on inner nodes, update to write when necessary

› Key challenge:
  – pointer indirection
  – concurrency memory model

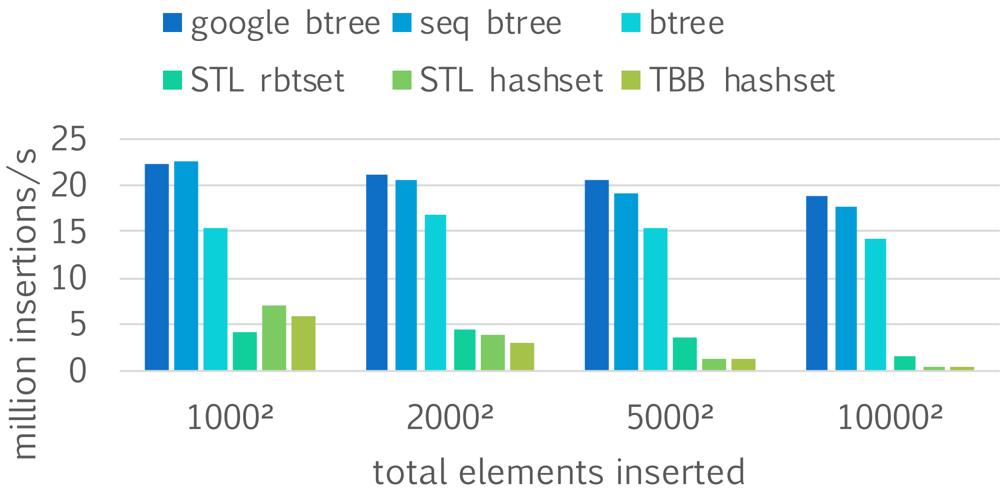


# B-tree Locking Strategies (cont)

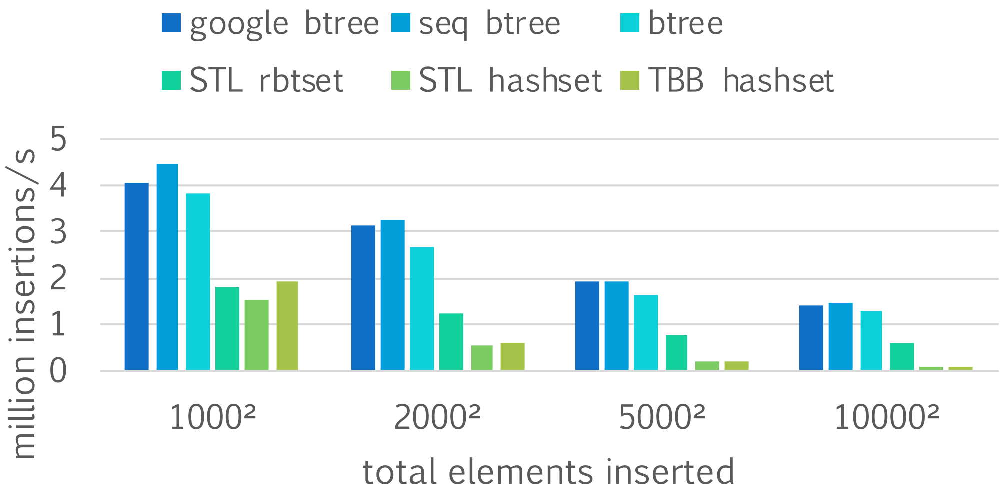



random order, on 4x8 core Intel Xeon E5-4650

# Sequential Performance



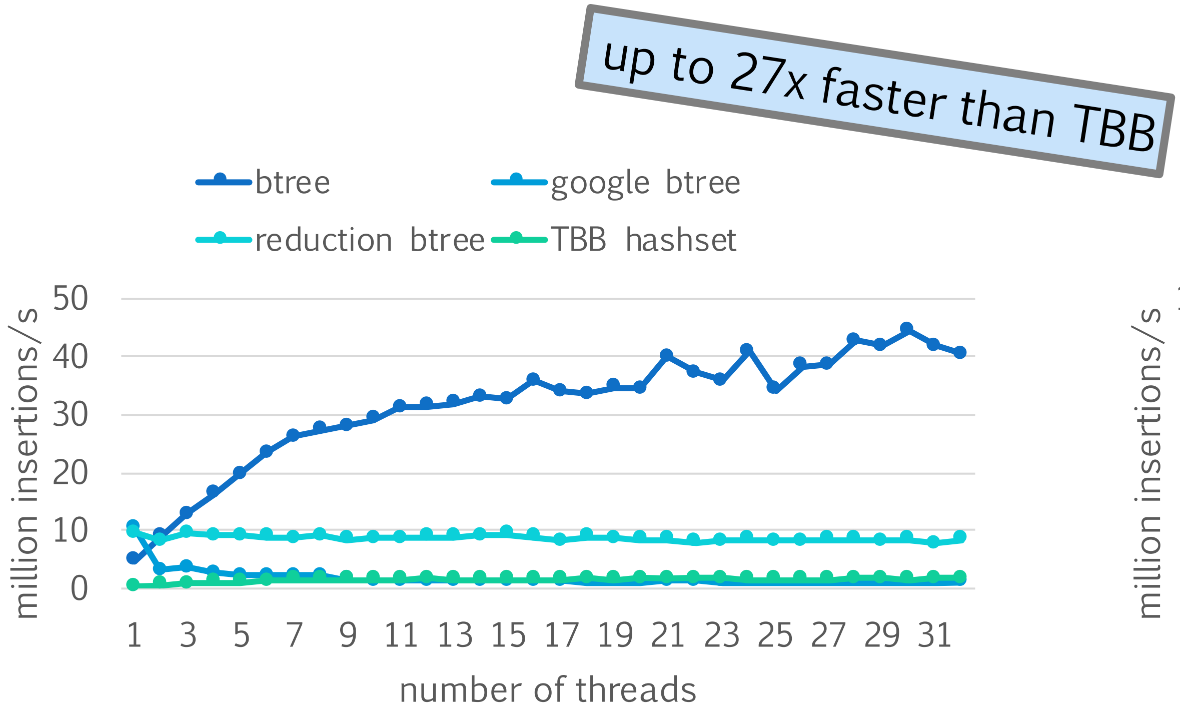

ordered insertion



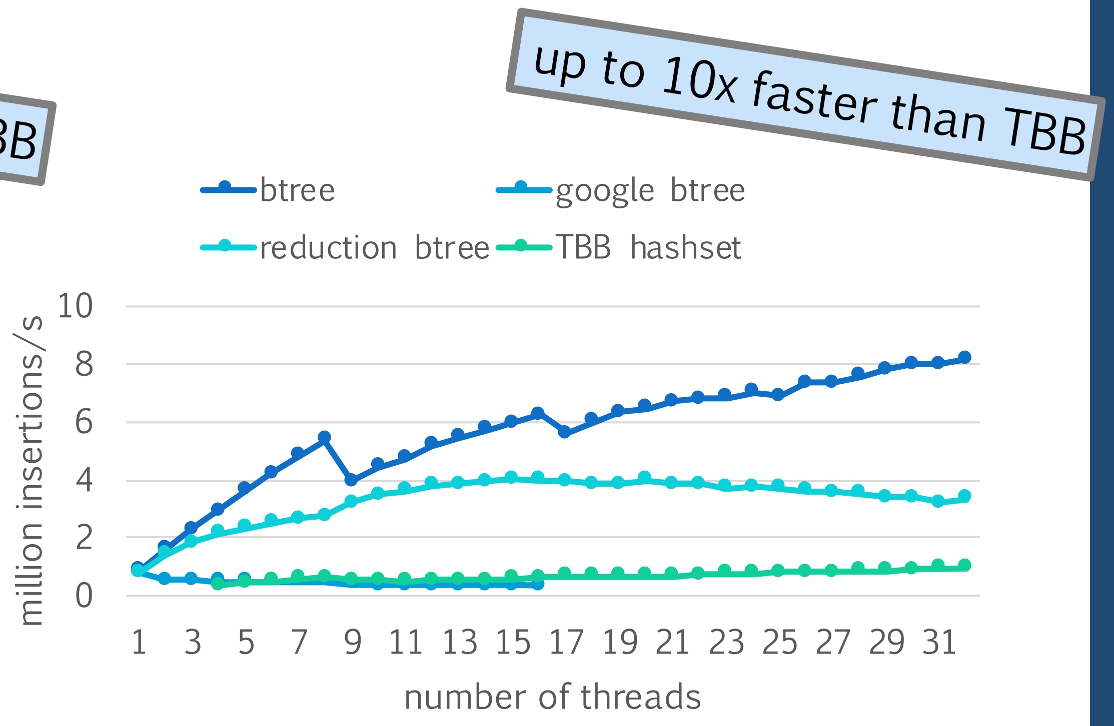

random order insertion

(additional data structures covered in paper)

# Parallel Performance



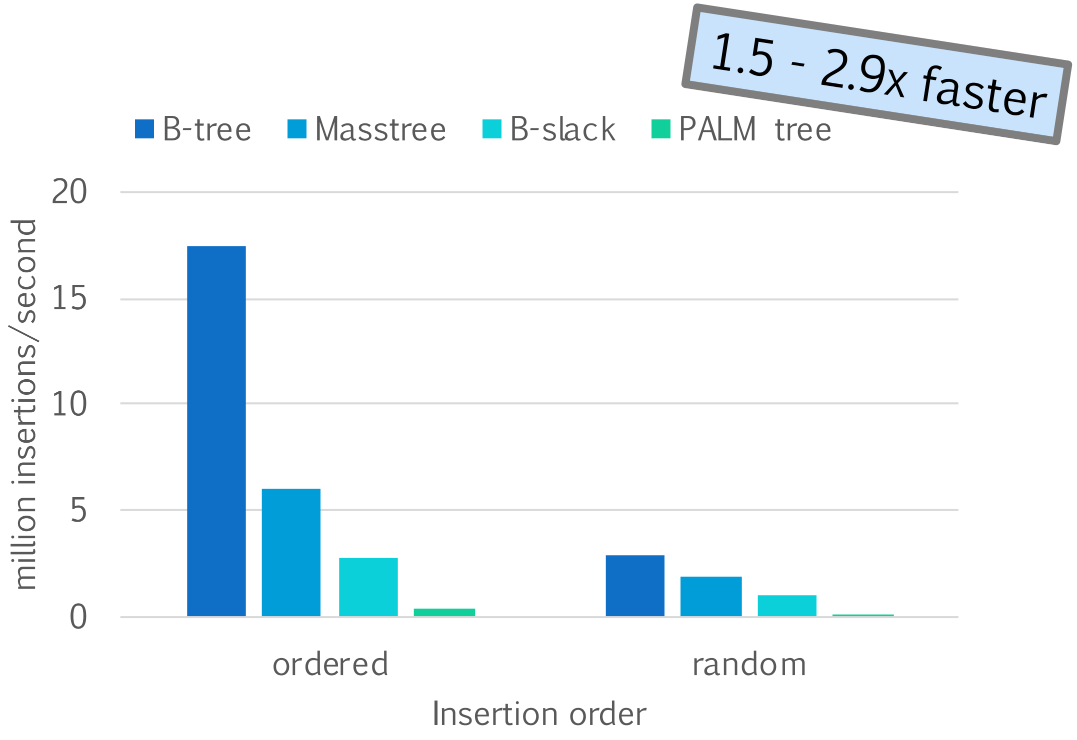

ordered insertion



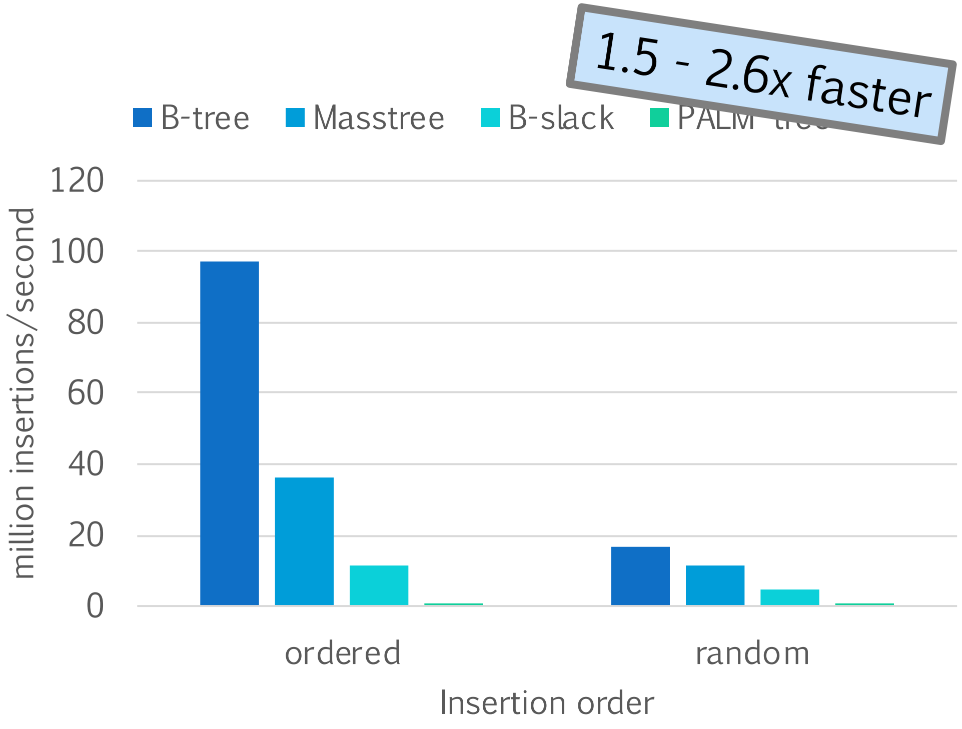

random order insertion

4x8 core Intel Xeon E5-4650

# Other Concurrent Tree Data Structures

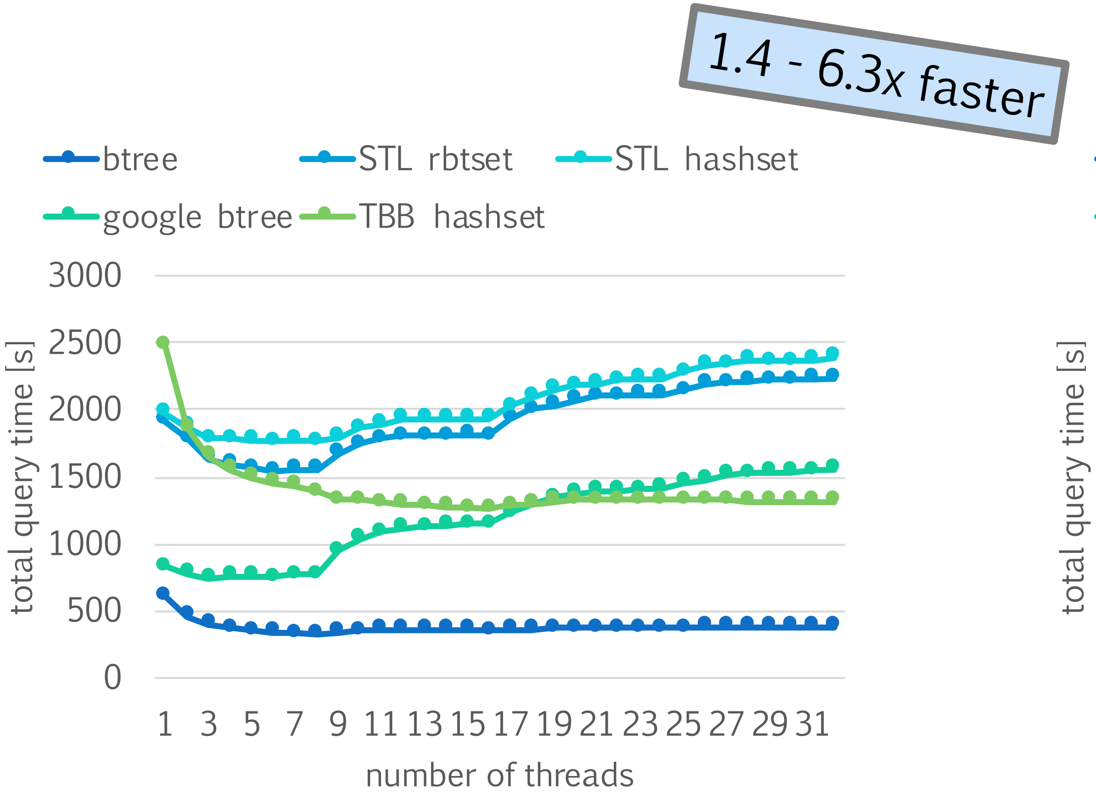



single-threaded

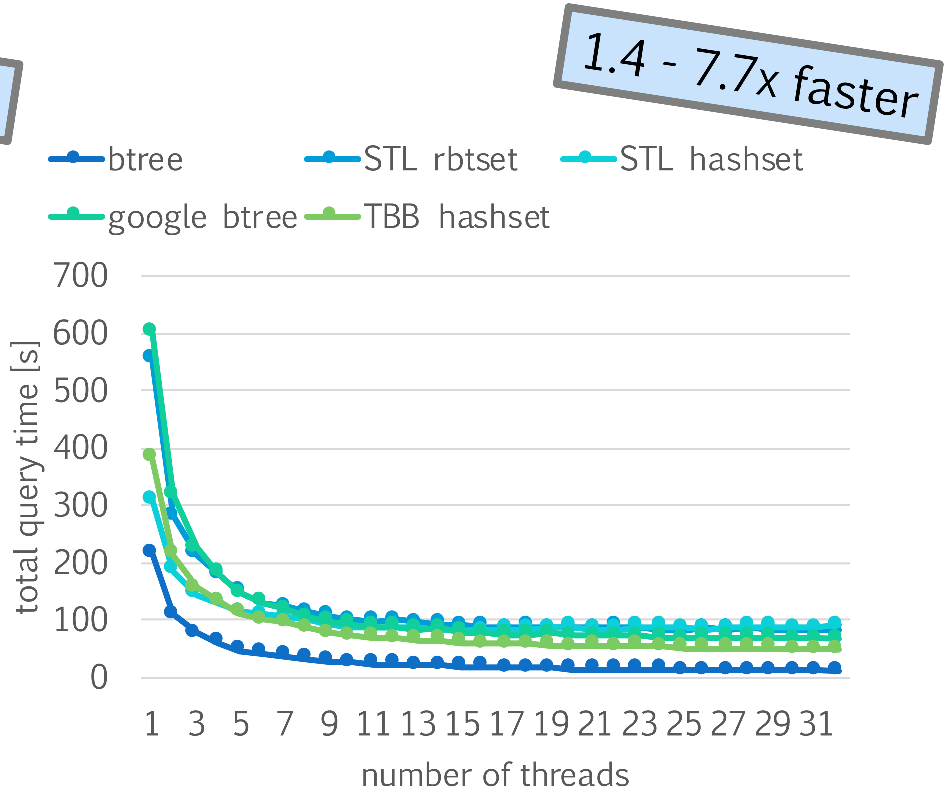



8 threads

# Datalog Query Processing




context sensitive
var-points-to analysis




security vulnerability
analysis

# Conclusion

› Developed concurrent set for Datalog relations:

- B-tree foundation
  › good sequential performance, cache friendly
- Fine-grained synchronization
  › based on customized seqlock variant

› Results:

- up to **59x** faster than state-of-the-art **hash** based **sets**
- up to **2.9x** faster than state-of-the-art **tree** based **sets**
- up to **7.7x** faster for real-world **query processing**

› Future work:

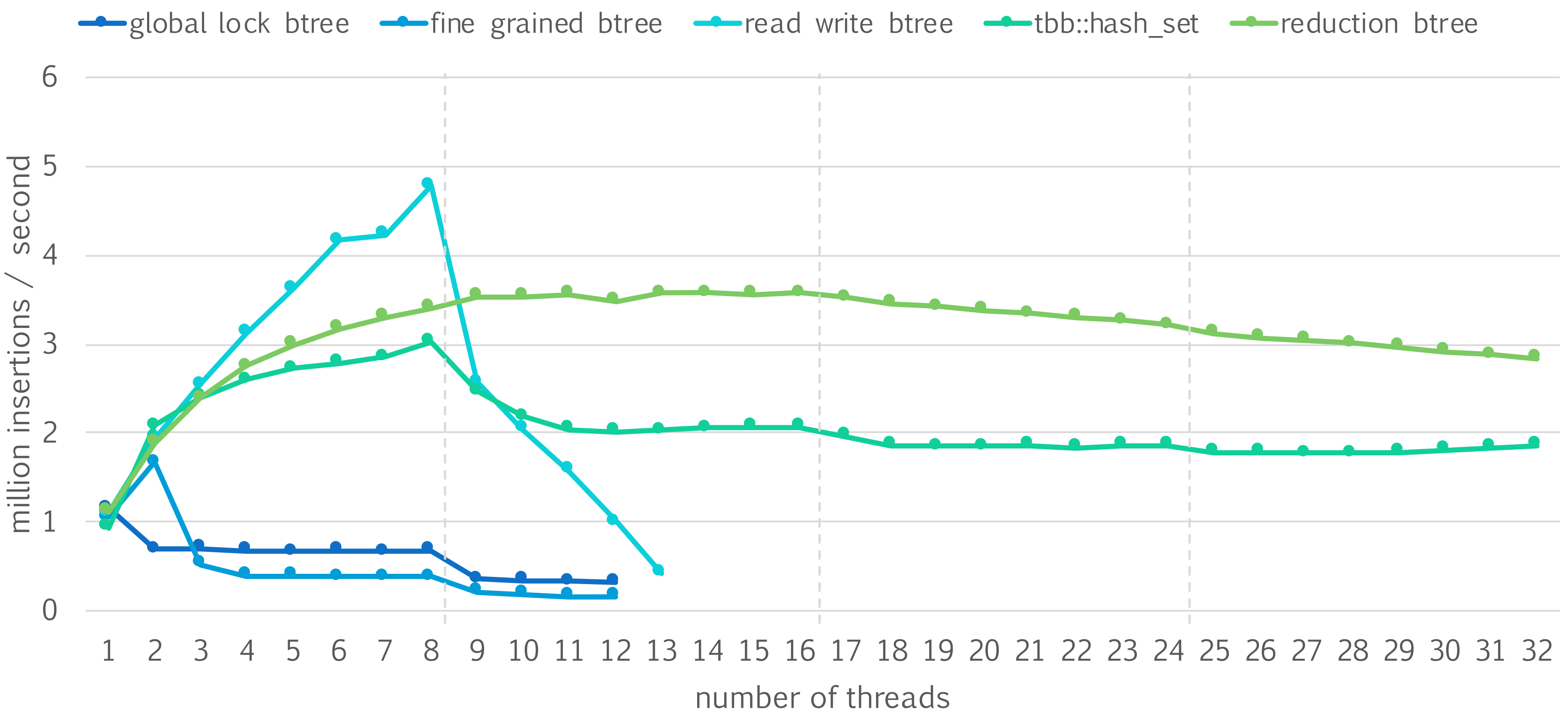
- investigate other data structures for specialized use cases

# Thank you!

visit us on https://souffle-lang.github.io

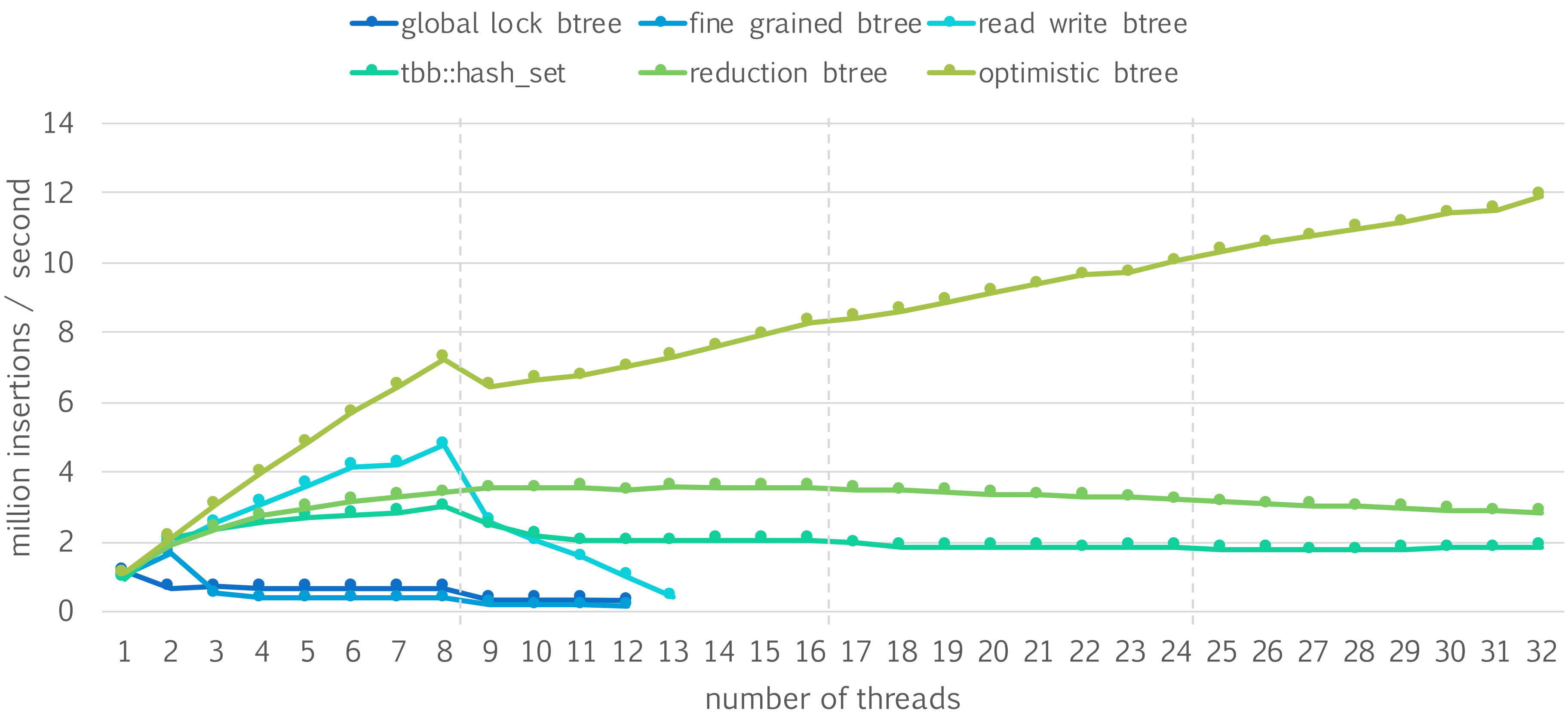
sources: https://github.com/souffle-lang/souffle

# B-tree Locking Strategies




random order, on 4x8 core Intel Xeon E5-4650

# B-tree Locking Strategies (cont)




random order, on 4x8 core Intel Xeon E5-4650