The ART of **Practical** Synchronization

Viktor Leis, Florian Scheibner,
Alfons Kemper, Thomas Neumann

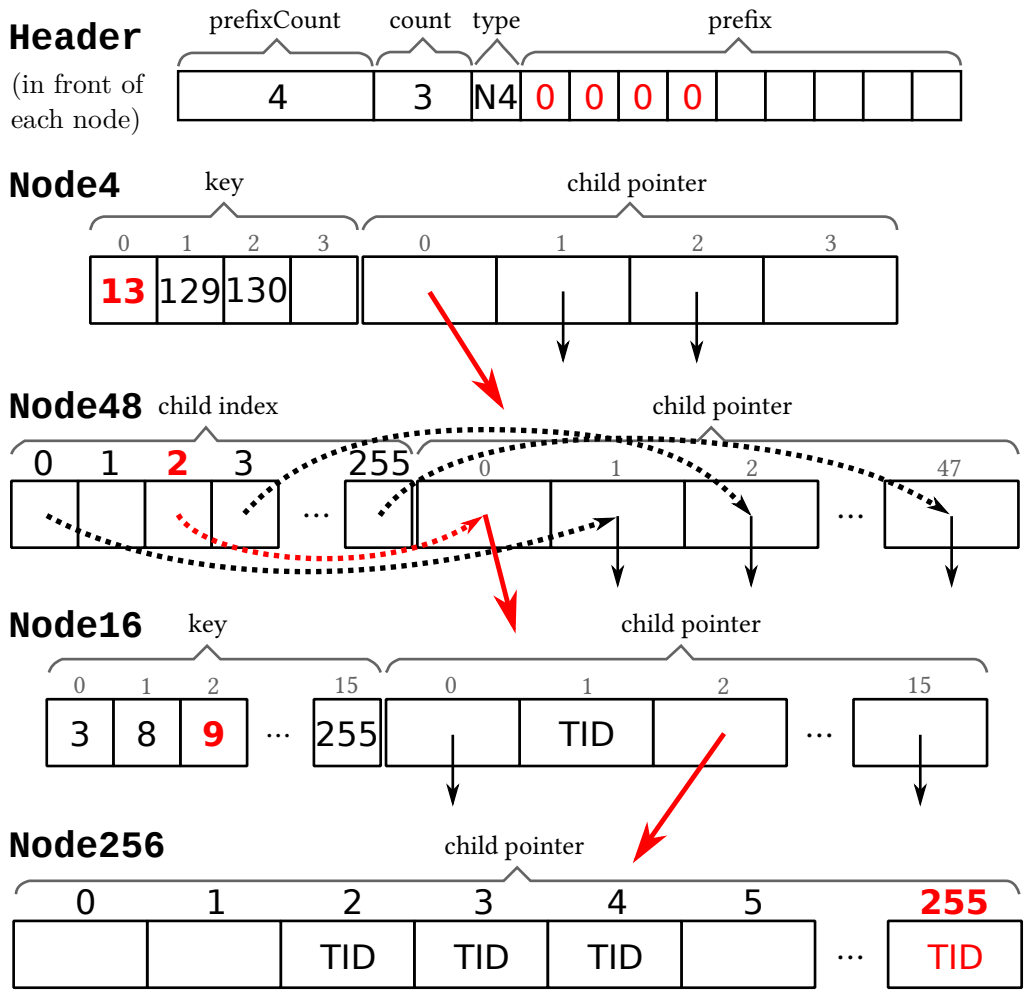
Technische Universität München

Introduction

- low-level synchronization ("latching") usually decides how well a workload scales on multi-core CPUs
- database systems have lots of data structures: index structures, tuple storage, job queues, buffer management data structures, etc.
- goal of this work: consolidate prior work and distill general principles for synchronizing data structures in general

Case Study: The Adaptive Radix Tree (ART)

- general-purpose, order-preserving index for in-memory DBMSs
- originally designed for high single-threaded performance
- until now, no efficient synchronized variant



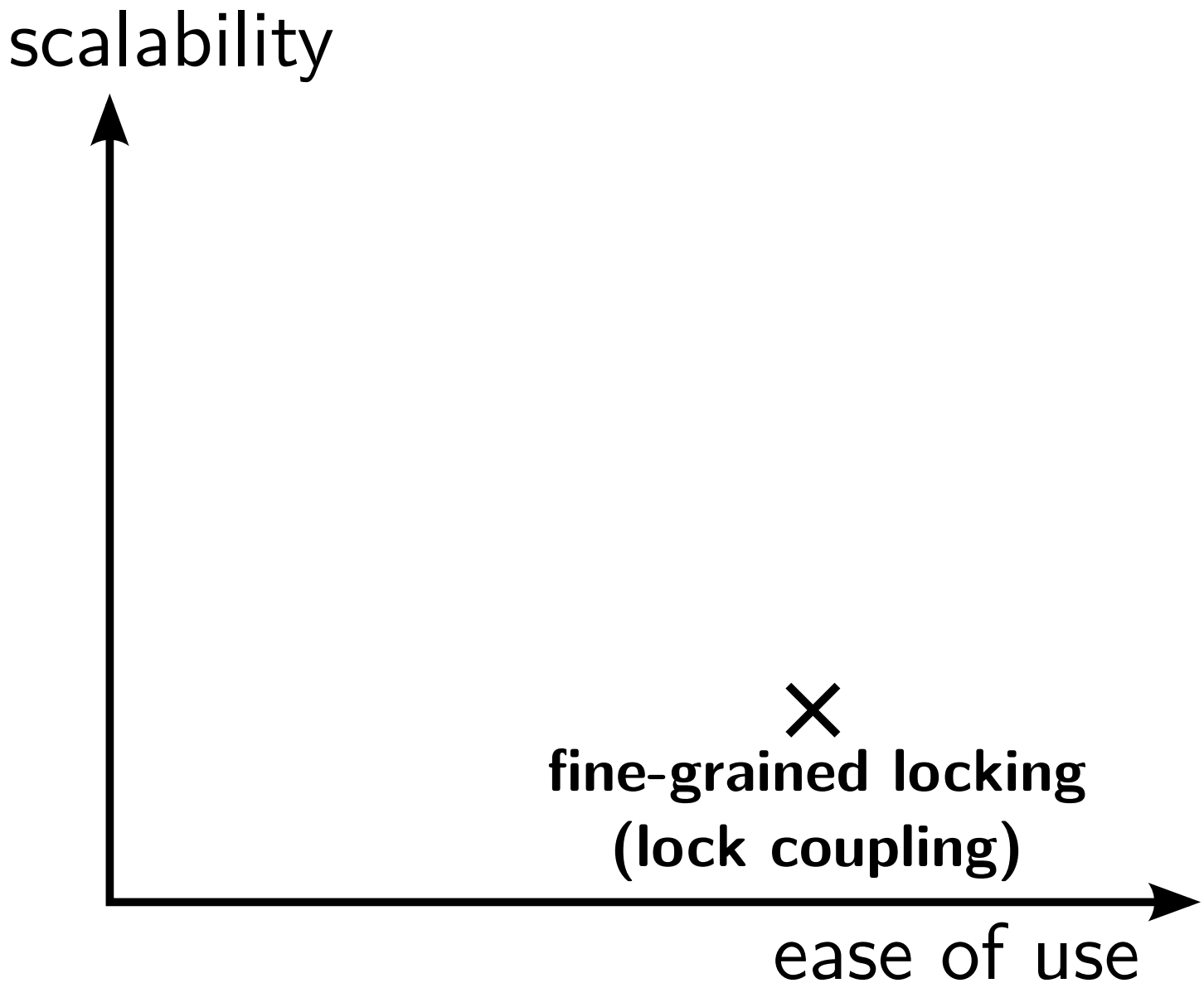
Lock Coupling for ART

- very easy to apply to ART
- modifications only change 1 node and (sometimes) its parent
- can use read/write locks to allow for more concurrency

Lock Coupling for ART

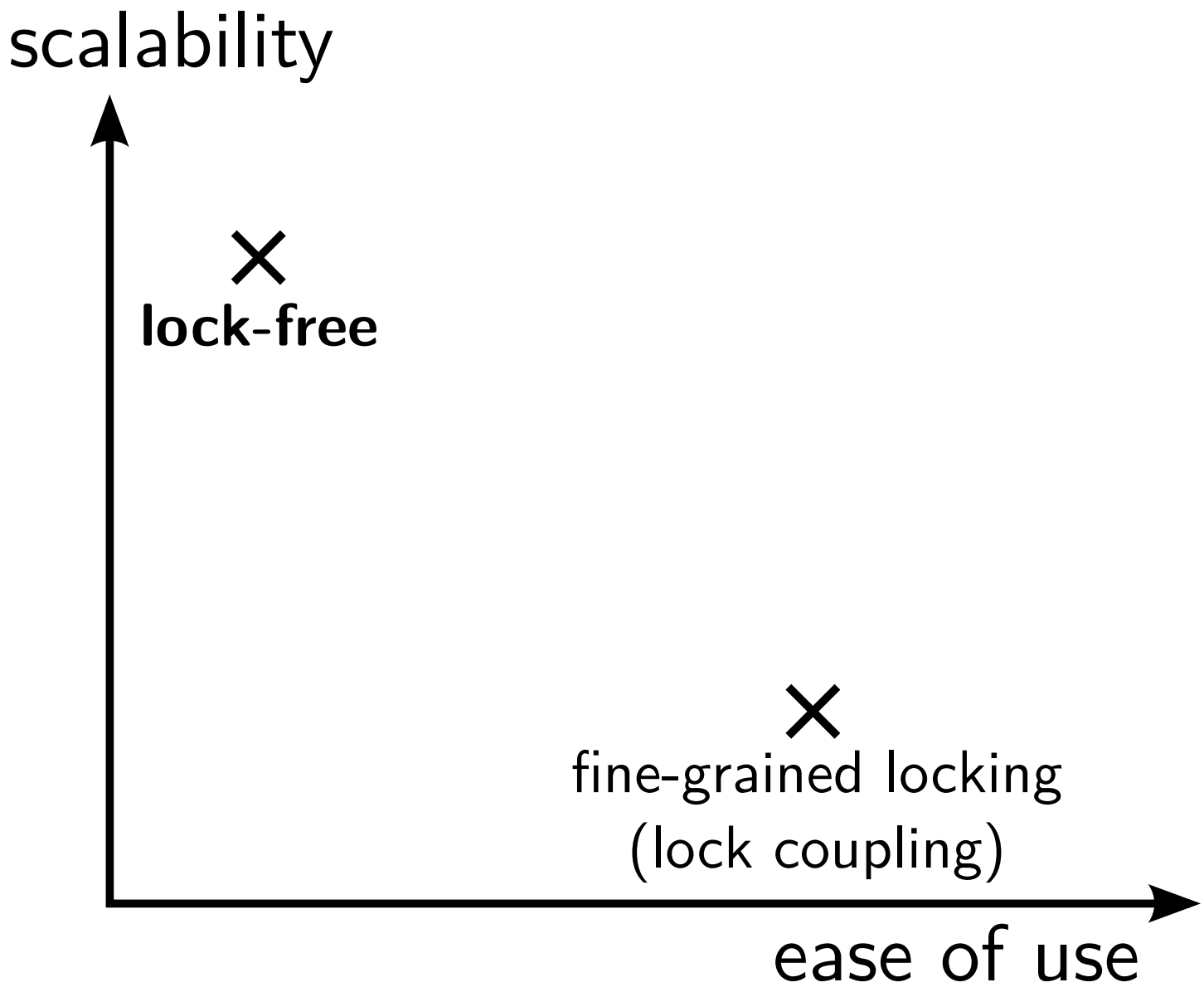
- very easy to apply to ART
- modifications only change 1 node and (sometimes) its parent
- can use read/write locks to allow for more concurrency

```
lookup(key, node, level, parent)
   readLock(node)
   if parent != null
      unlock(parent)
   // check if prefix matches, may increment level
   if !prefixMatches(node, key, level)
      unlock(node)
      return null  // key not found
   // find child
   nextNode = node.findChild(key[level])
   if isLeaf(nextNode)
      value = getLeafValue(nextNode)
      unlock(node)
      return value  // key found
   if nextNode == null
      unlock(node)
      return null  // key not found
   // recurse to next level
   return
```


scalability
fine-grained locking
(lock coupling)
ease of use

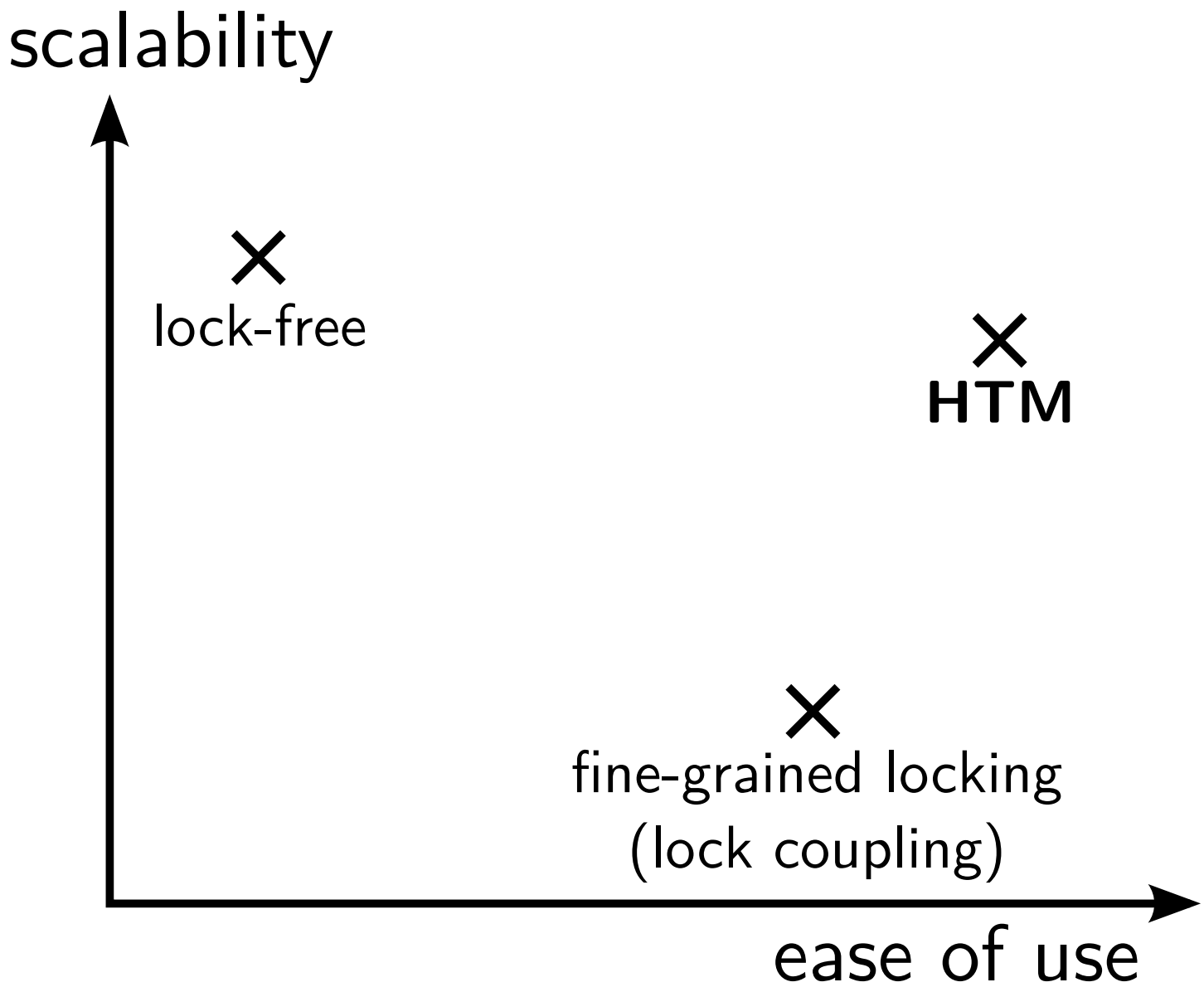
Lock-free ART?

- non-blocking data structures are extremely difficult to design, to implement, and to debug
- every non-trivial lock-free data structure is a research contribution
- non-blocking data structures add significant overhead
  - Bw-tree: extra delta records in front of each node
  - Split-ordered list (state-of-the-art lock-free hash table): dummy nodes
- a hypothetical lock-free ART variant would
  - require significant changes to the data structure (path compression is a major issue)
  - likely be slower than the methods presented in the following


scalability
lock-free
fine-grained locking
(lock coupling)
ease of use

Hardware Transactional Memory (HTM)

+ very easy to use (with coarse-grained, elided locks)
+ often scales well
– requires special (not yet widespread) hardware support
– sometimes hard to predict/debug behavior


scalability
lock-free
HTM
fine-grained locking
(lock coupling)
ease of use

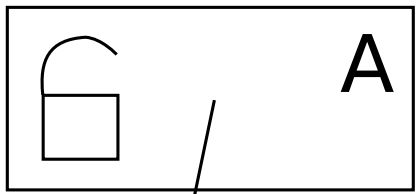
Optimistic Lock Coupling (1)

- add lock and version to each node
- write:
  - acquire lock (exclude other writers)
  - increment version when unlocking
  - do not acquire locks for nodes that are not modified (traverse like a reader)
- read:
  - do not acquire locks, proceed optimistically
  - detect concurrent modifications through versions (and restart if necessary)
  - can track changes across multiple nodes (lock coupling)

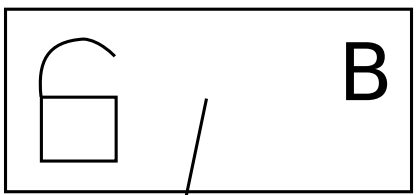
Optimistic Lock Coupling (2)

traditional **optimistic**

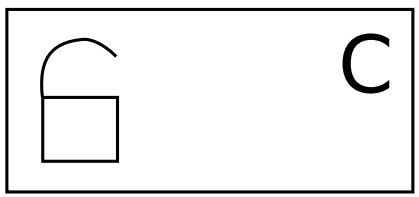
1. lock node A
2. search node A



3. lock node B
4. unlock node A
5. search node B



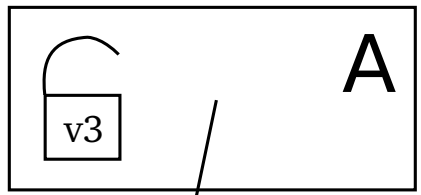
6. lock node C
7. unlock node B
8. search node C
9. unlock node B



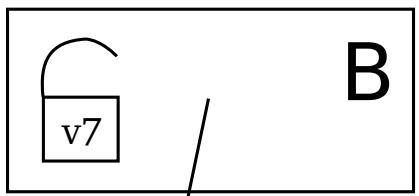
Optimistic Lock Coupling (2)

traditional **optimistic**

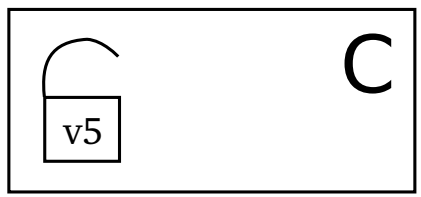
1. lock node A
2. search node A



3. lock node B
4. unlock node A
5. search node B



6. lock node C
7. unlock node B
8. search node C
9. unlock node B



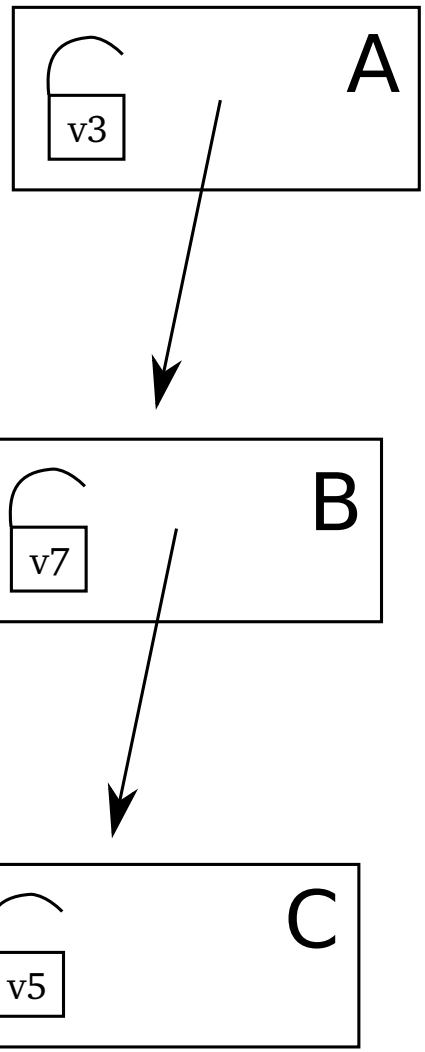
Optimistic Lock Coupling (2)

traditional

1. lock node A
2. search node A

3. lock node B
4. unlock node A
5. search node B

6. lock node C
7. unlock node B
8. search node C
9. unlock node B



optimistic

1. read version v3
2. search node A

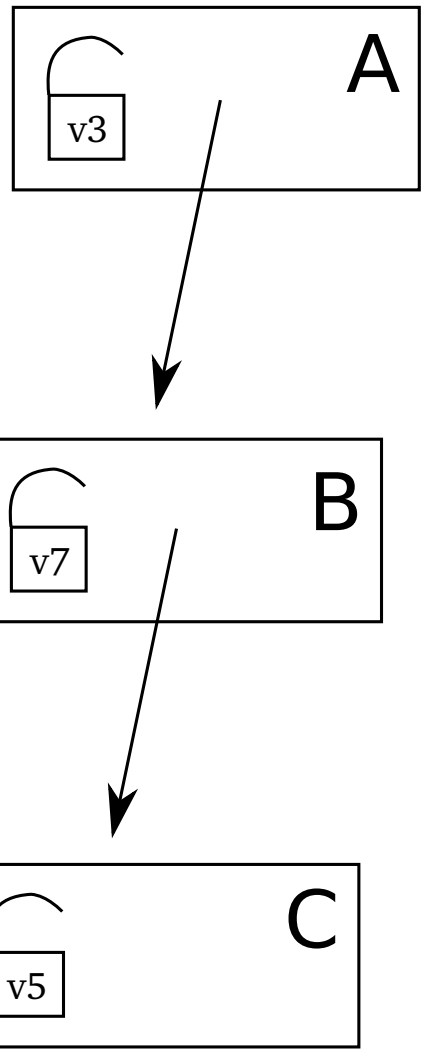
Optimistic Lock Coupling (2)

traditional

1. lock node A
2. search node A

3. lock node B
4. unlock node A
5. search node B

6. lock node C
7. unlock node B
8. search node C
9. unlock node B



optimistic

1. read version v3
2. search node A

3. read version v7
4. re-check version v3
5. search node B

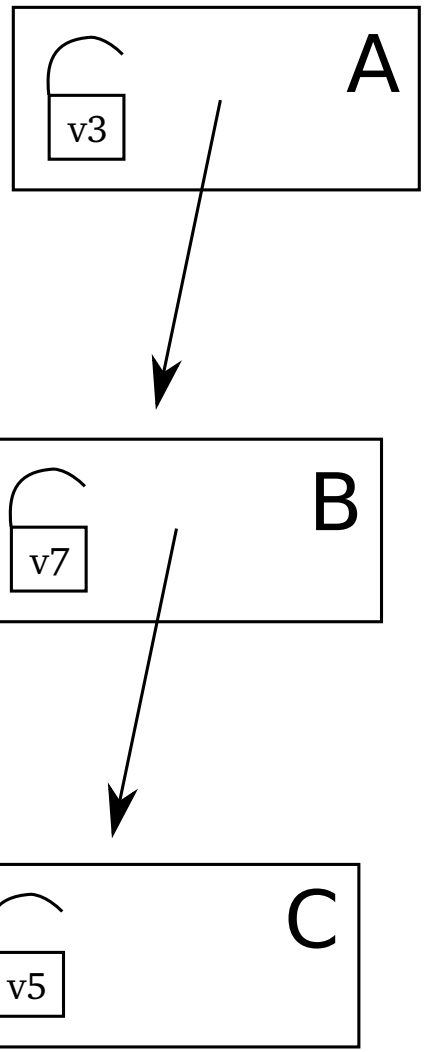
Optimistic Lock Coupling (2)

traditional

1. lock node A
2. search node A

3. lock node B
4. unlock node A
5. search node B

6. lock node C
7. unlock node B
8. search node C
9. unlock node B



optimistic

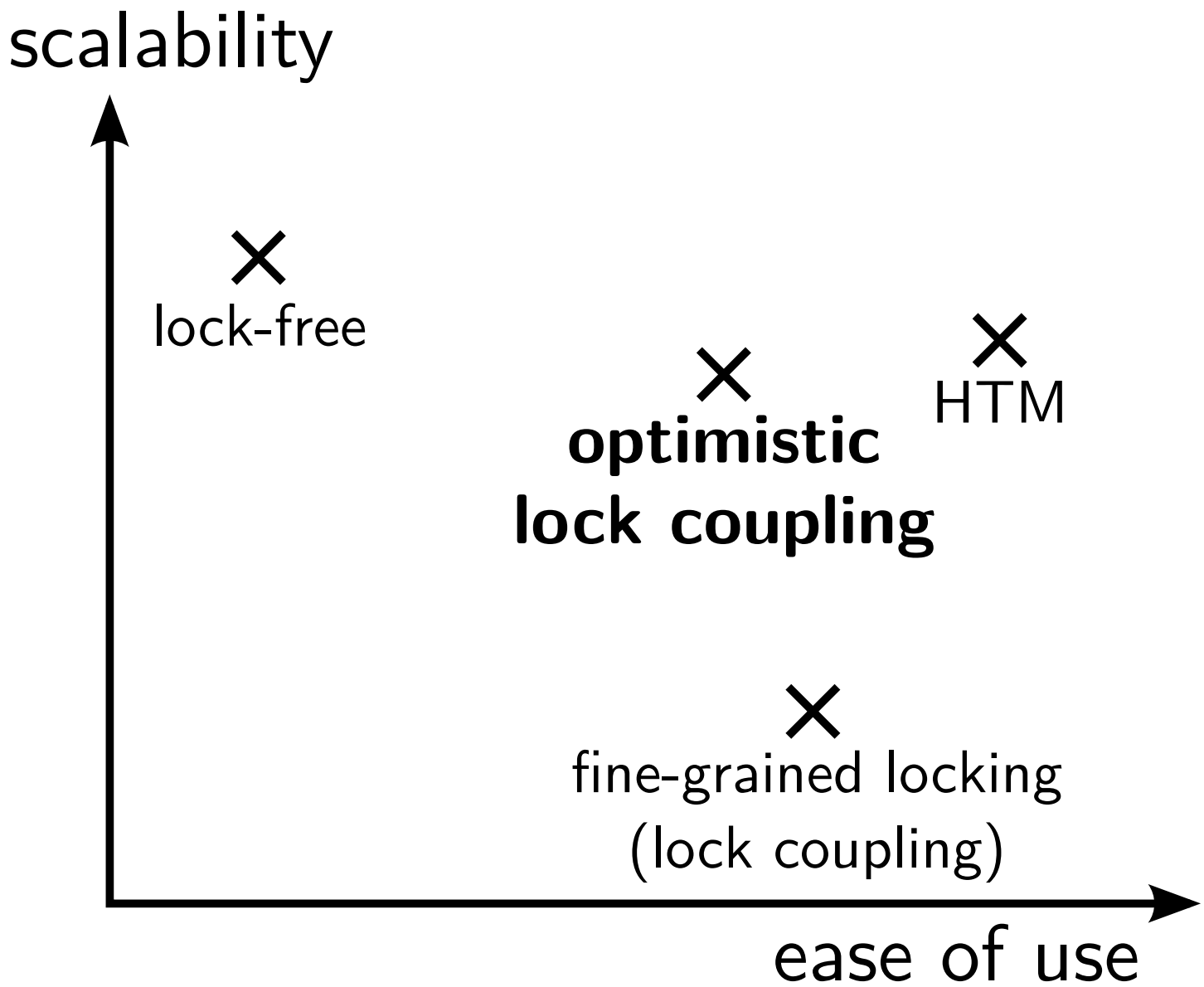
1. read version v3
2. search node A

3. read version v7
4. re-check version v3
5. search node B

6. read version v5
7. re-check version v7
8. search node C
9. re-check version v5

Optimistic Lock Coupling (3)

+ can easily be applied to most data structures (no modifications necessary)
+ scales well
+ low overhead
− can lead to (unnecessary) aborts


scalability
lock-free
optimistic
lock coupling
HTM
fine-grained locking
(lock coupling)
ease of use

Read-Optimized Write EXclusion (ROWEX) (1)

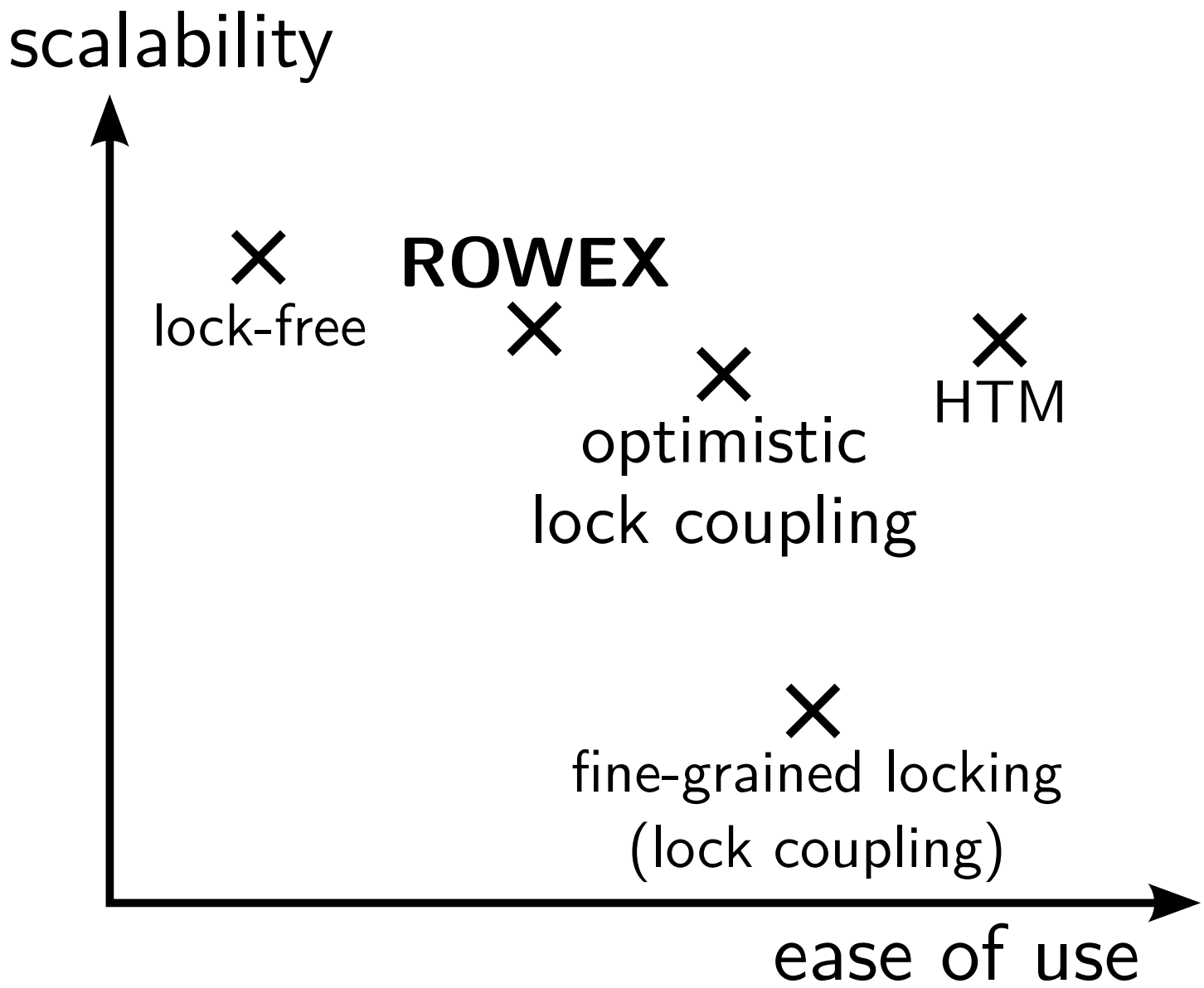
- add lock to each node
- write:
  - acquire lock (excludes writers)
  - make sure than any modification leaves the tree in a state safe for readers
- read:
  - simply proceed without observing locks or versions

Read-Optimized Write EXclusion (ROWEX) (2)

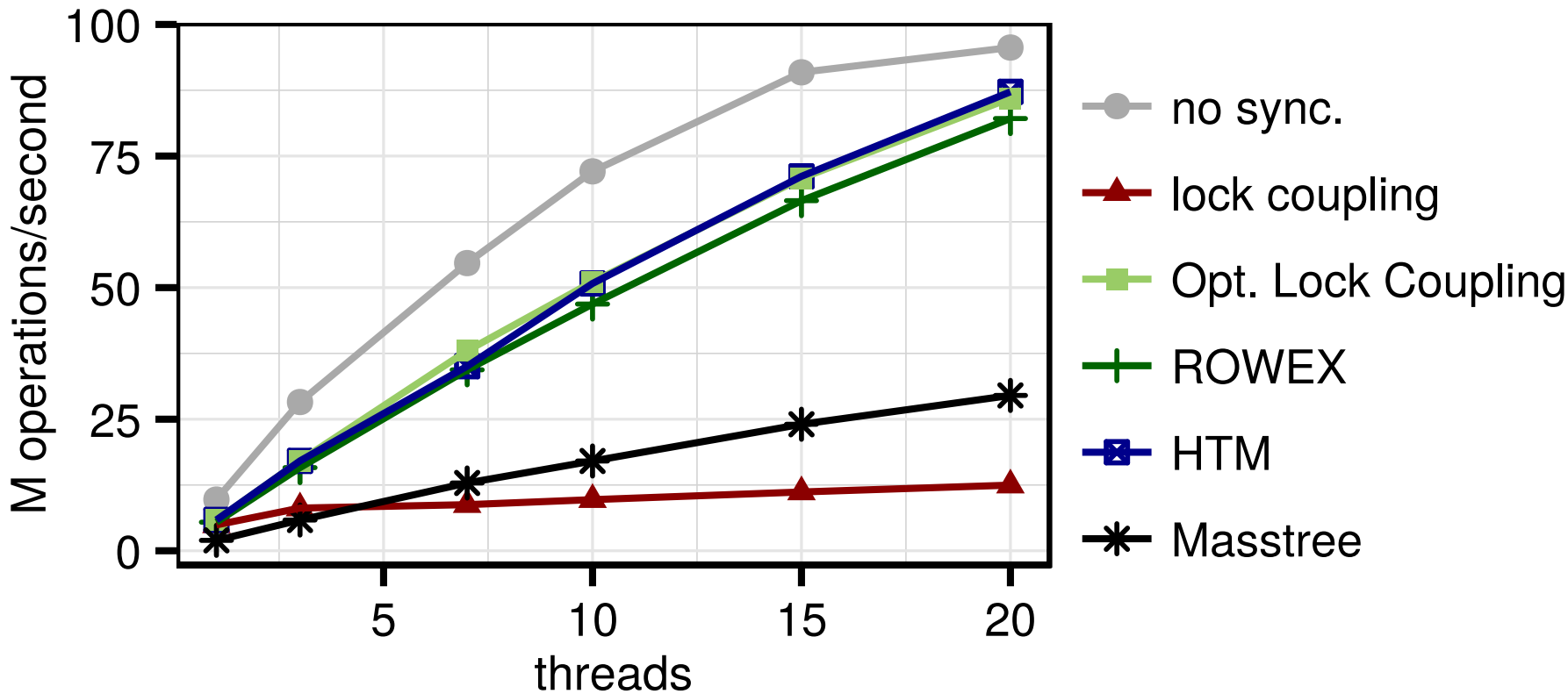
+ scales well
+ reads are non-blocking (always successful and there are no restarts)
+ easier to implement than lock-free data structures
– more difficult to implement than Optimistic Lock Coupling (requires modifications to the underlying data structure)

Synchronizing ART with ROWEX

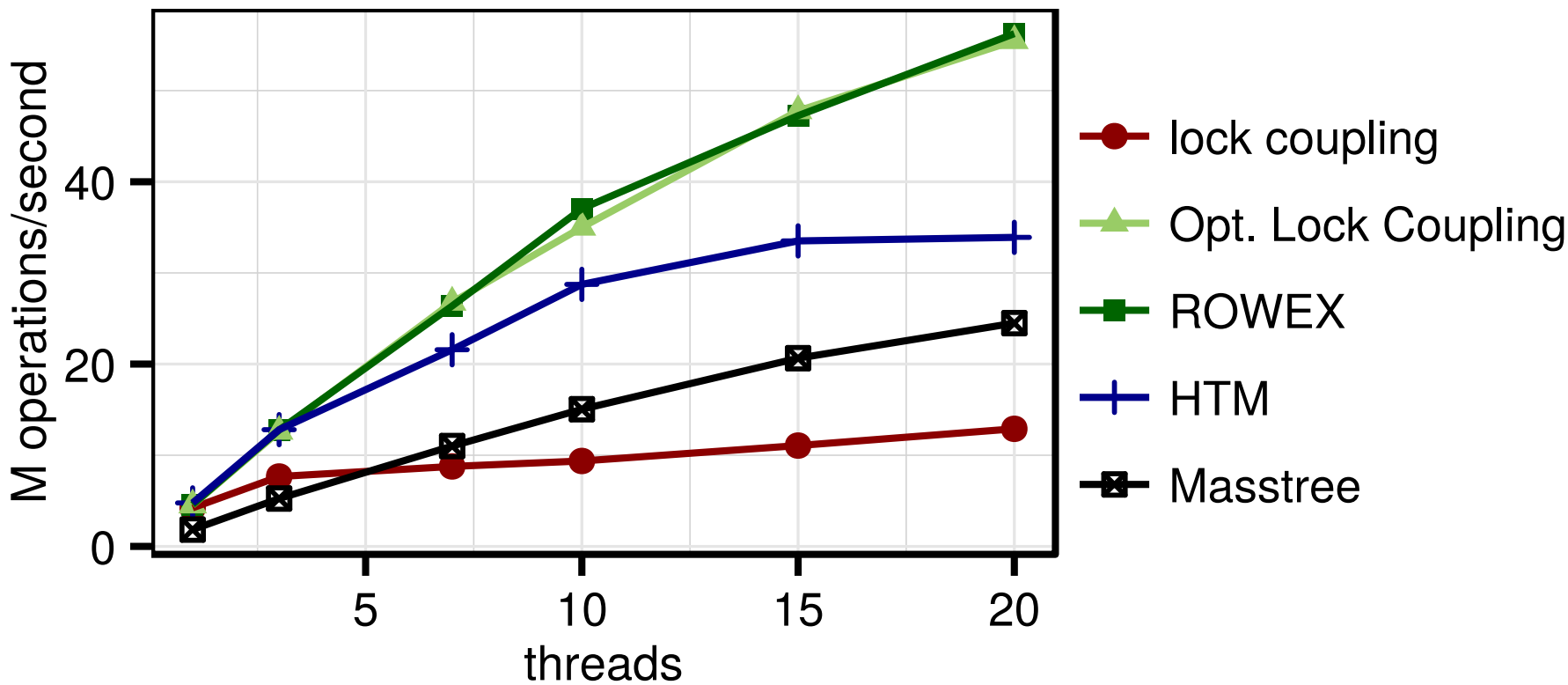
- local modifications:
  - make key and child pointer accesses atomic (`std::atomic`)
  - make `Node4` and `Node16` unsorted and append-only
- grow/shrink a node:
  - lock node and its parent
  - create new node and copy entries
  - set parent pointer to the new node
- path compression:
  - modify prefix atomically
  - add `level` field to each node


scalability
ROWEX
lock-free
optimistic
lock coupling
HTM
fine-grained locking
(lock coupling)
ease of use

Lookup (50M 8B Integers)



Insert (50M 8B Integers)

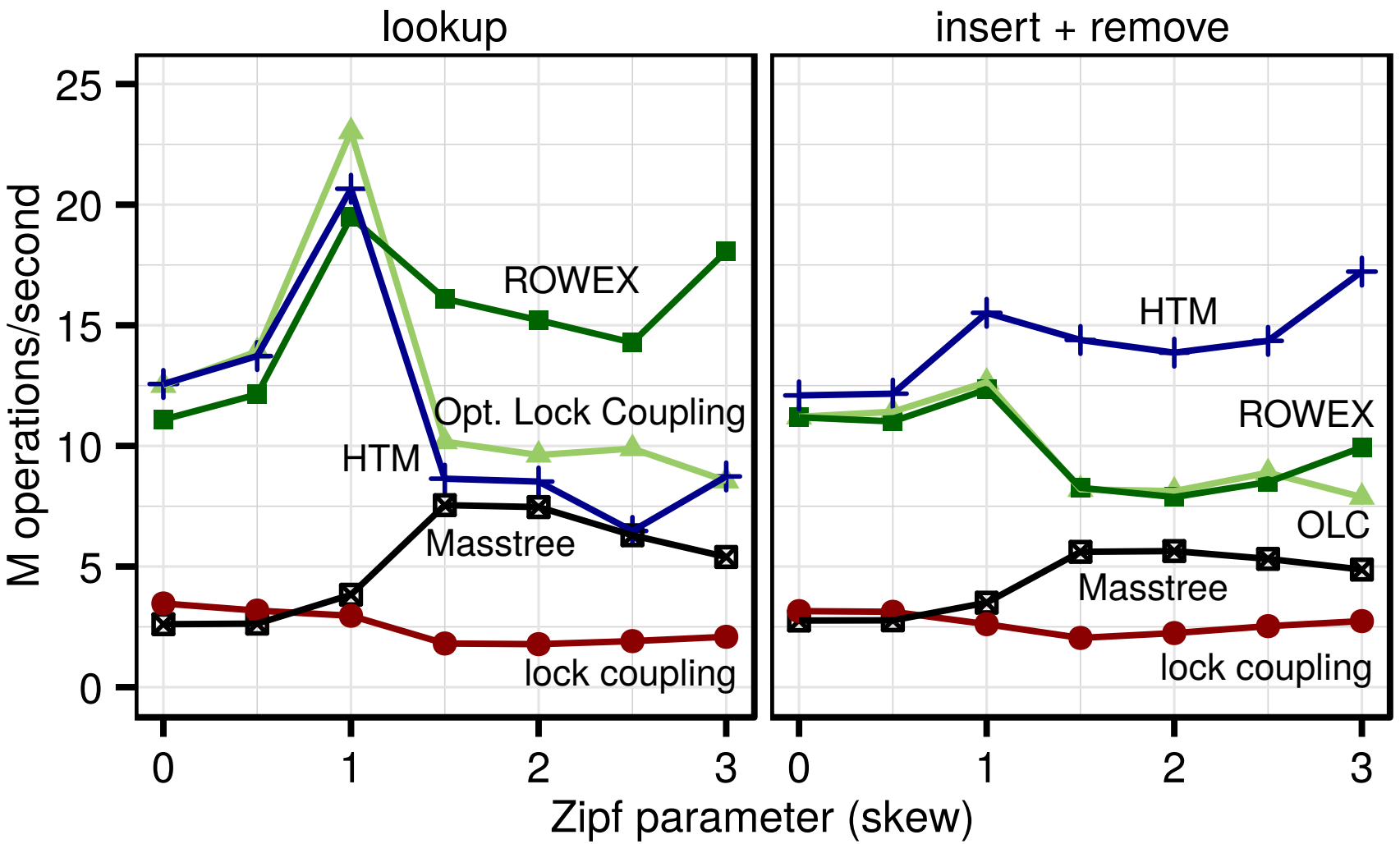


Summary

- traditional fine-grained locking does not scale for tree-like index structures
- locks are fine if they are only acquired by writers and only on nodes that are modified
- Optimistic Lock Coupling and ROWEX are two highly practical alternatives to the lock-free paradigm

`https://github.com/flode/ARTSynchronized`

Contention


lookup
insert + remove
M operations/second
25
20
15
10
5
0
ROWEX
Opt. Lock Coupling
HTM
Masstree
lock coupling
HTM
ROWEX
OLC
Masstree
lock coupling
0
1
2
3
Zipf parameter (skew)