Performance Analysis of Concurrent Red-Black Trees on HTM Platforms

D. Siakavaras K. Nikas G. Goumas N. Koziris

National Technical University of Athens
School of Electrical and Computer Engineering
Computing Systems Laboratory (CSLab)

TRANSACT 2015

Outline

Motivation

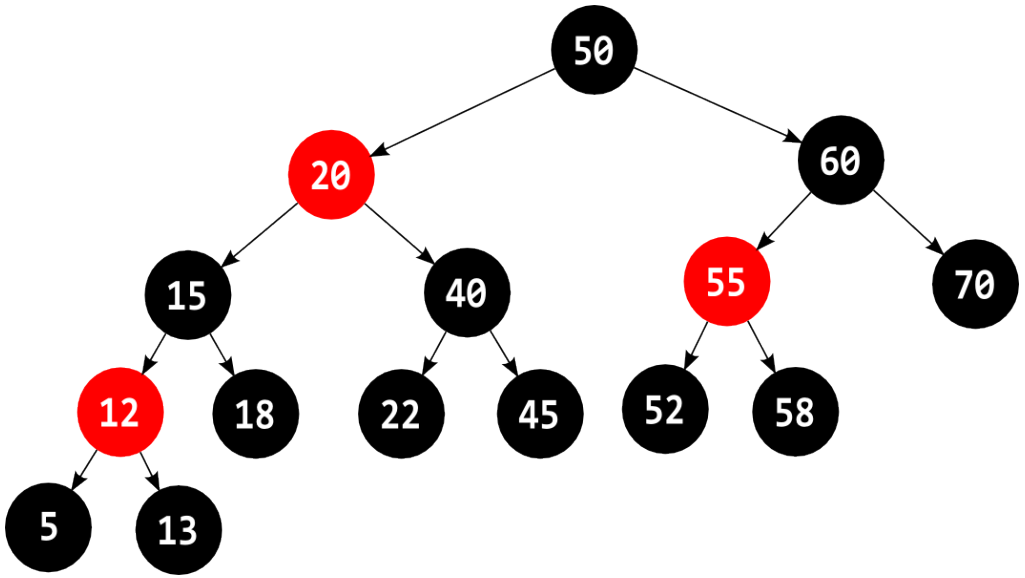
- Hardware Transactional Memory (HTM) support on modern processors:
  - IBM Power8, Blue Gene/Q, zEC12.
  - Intel Haswell, Broadwell ...
- We need to evaluate its performance with real life applications.

Red-Black trees:

- Widely used for dictionary implementations.
- Challenging to devise efficient concurrent implementations(e.g. with locks or atomic primitive).
- Their properties favor the usage of HTM.

Red-Black Trees

Red-Black Trees in a nutshell



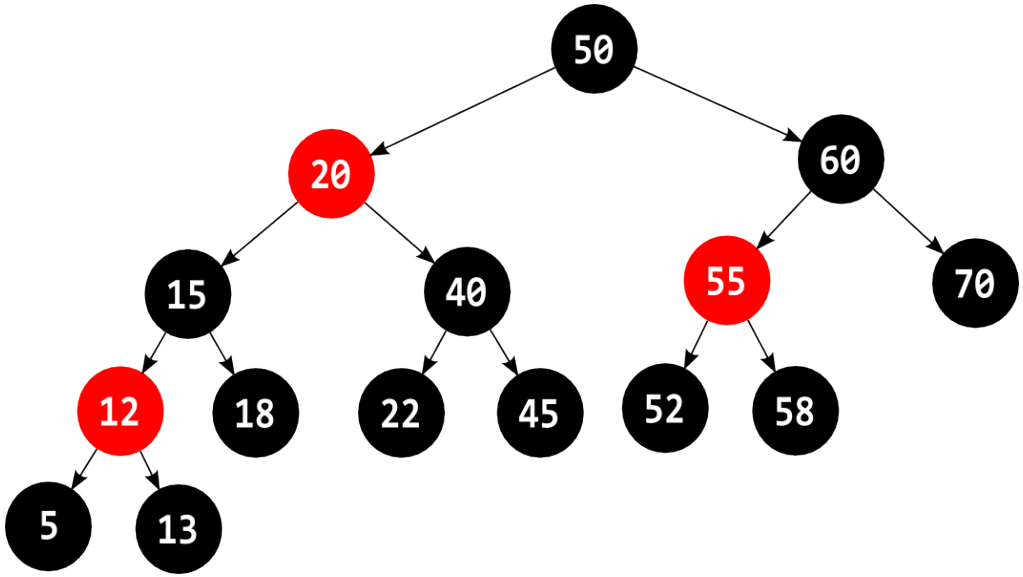
Definition

Binary Search Tree (BST) with the following additional properties:

1. A node is either **black** or **red**.
2. The root is **black**.
3. All leaves are **black**.
4. Every **red** node must have two **black** children.
5. Every path from a given node to any of its descendant leaves contains the same number of **black** nodes.

The above properties guarantee that the tree is almost balanced.

Red-Black Trees in a nutshell



Applications

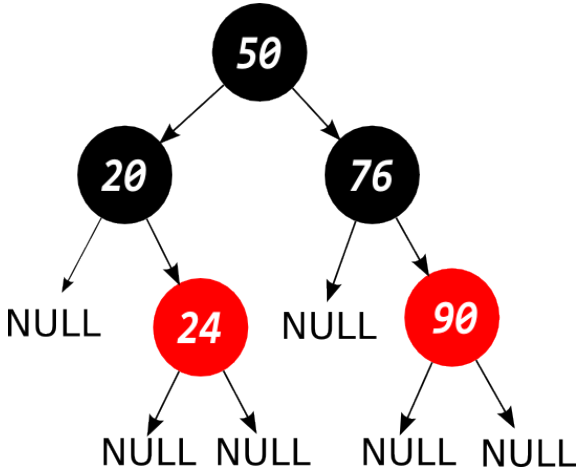
Dictionary ADT:

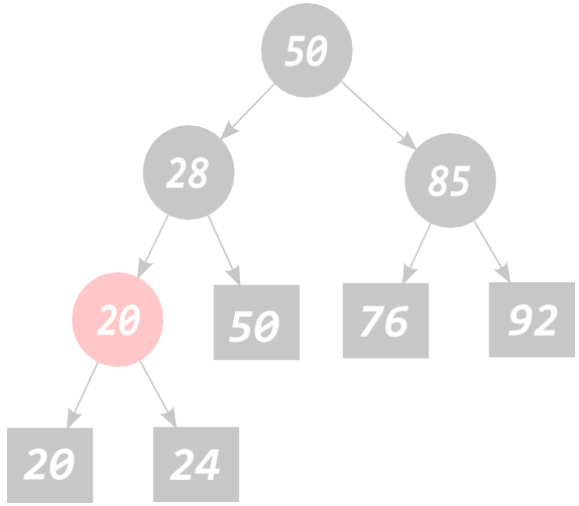
- Colection of *(key, value)* pairs.

Supports three operations:

- *Lookup*
- *Insert*
- *Delete*

Internal VS External RBTs





Internal:

- Both keys and values are stored in every node.

External:

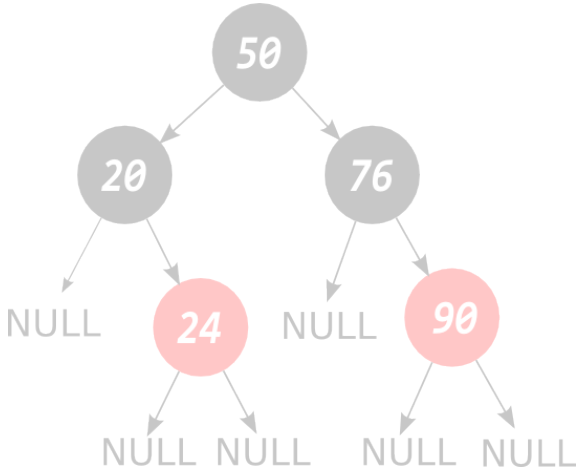
- Values are stored only in the leaves.
- Internal nodes are only used for routing purposes.

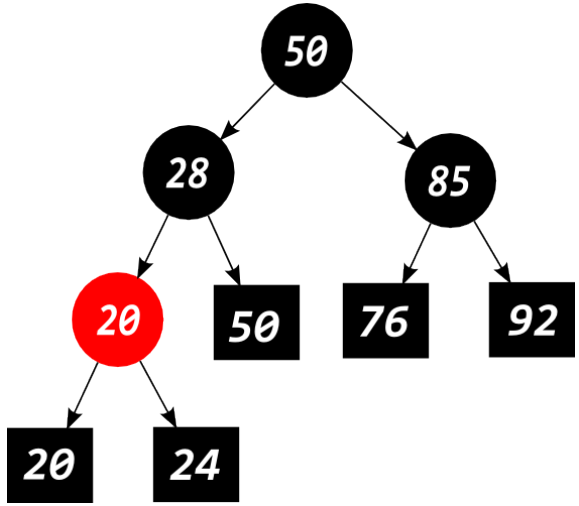
– Occupy more memory.

\+ Simplify delete operation.

- All our implementations are external trees.

Internal VS External RBTs





Internal:

- Both keys and values are stored in every node.

External:

- Values are stored only in the leaves.
- Internal nodes are only used for routing purposes.

– Occupy more memory.

\+ Simplify delete operation.

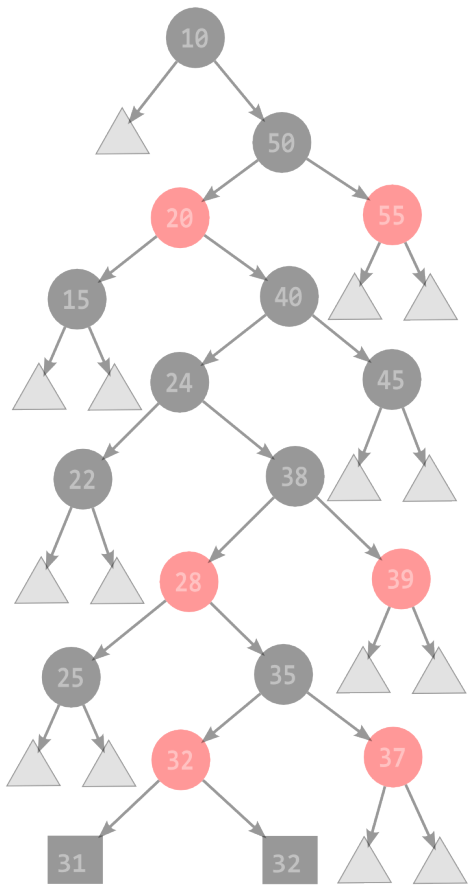
- All our implementations are external trees.

Red-Black tree operations: Lookup

Serial:

- Just like in BST.

Example: lookup(31)

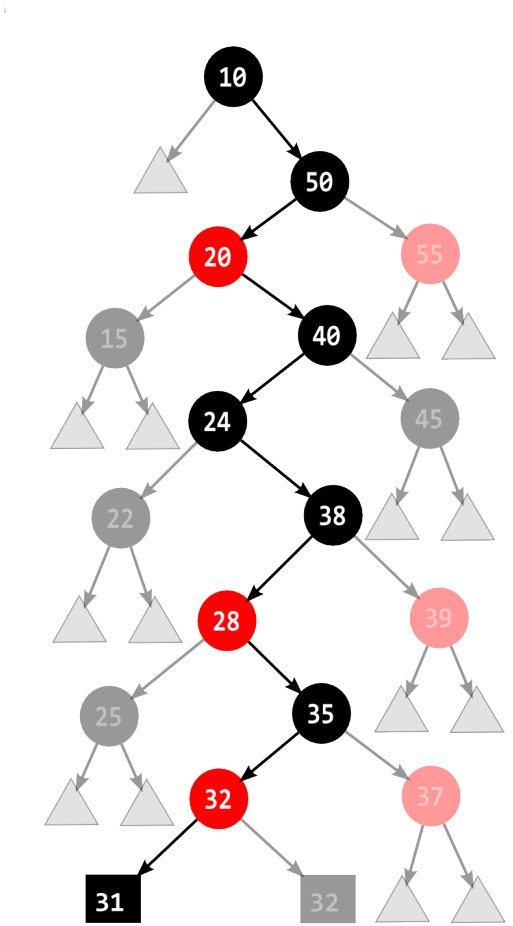


Red-Black tree operations: Lookup

Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Example: lookup(31)



Red-Black tree operations: Lookup

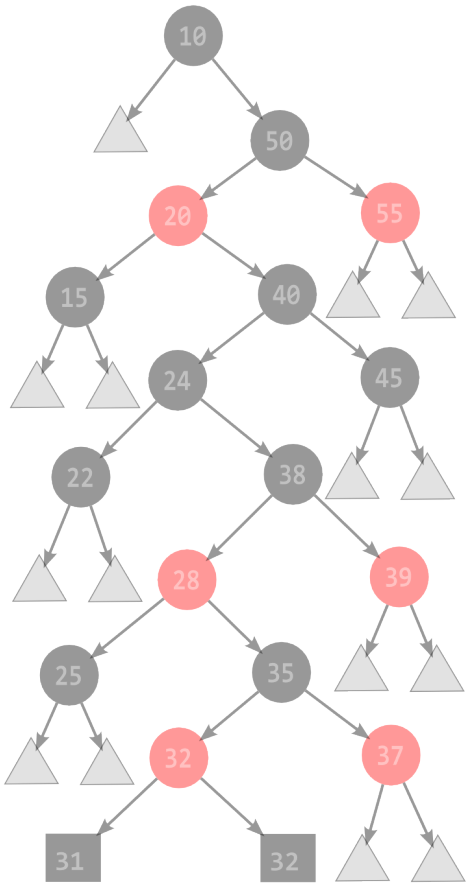
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

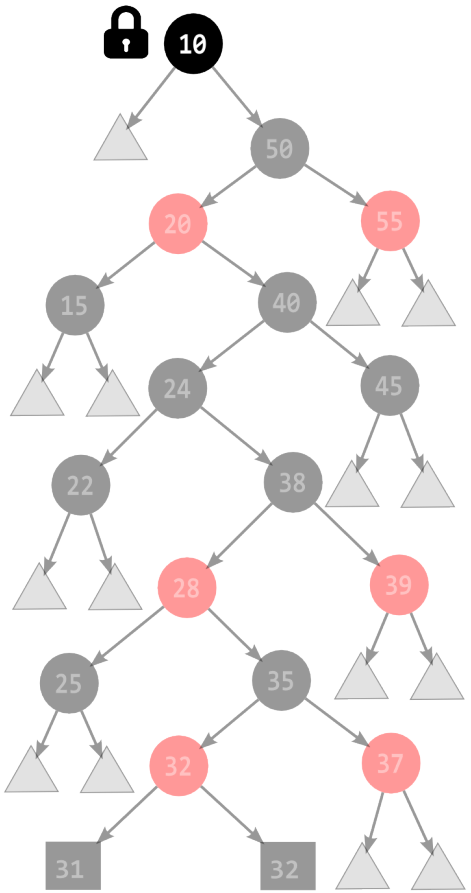
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

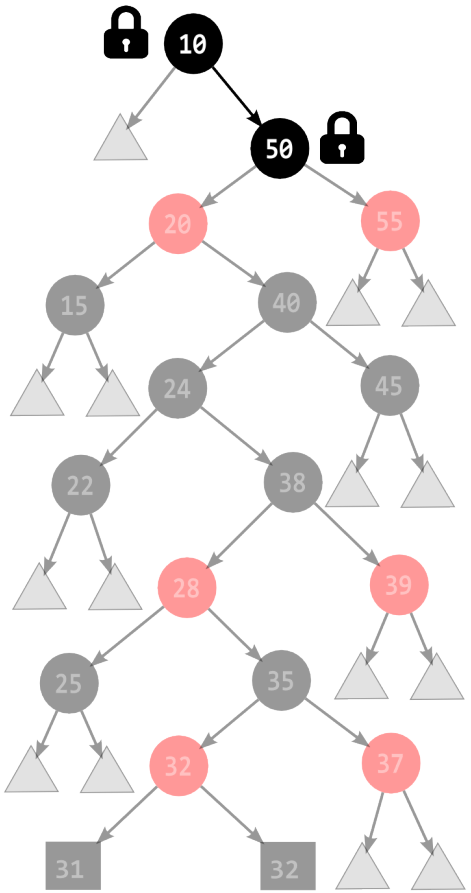
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

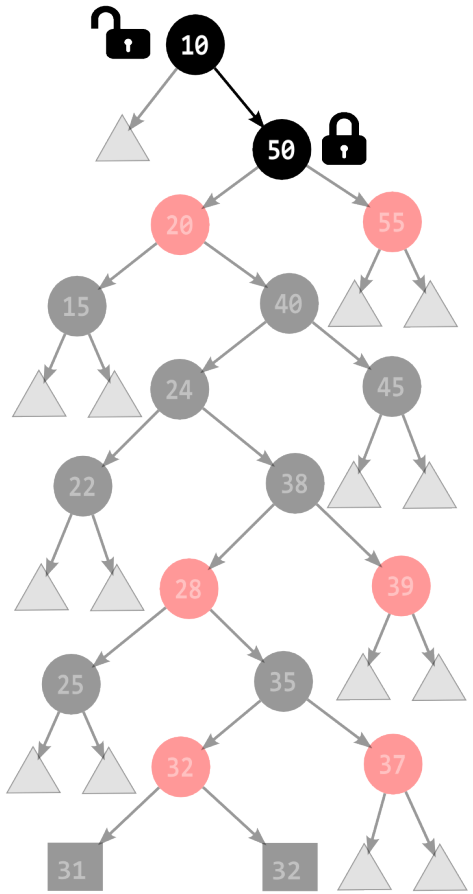
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

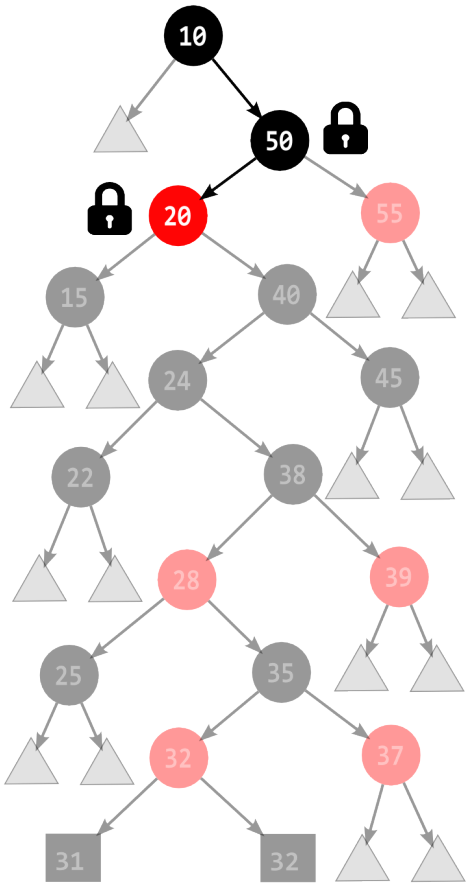
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

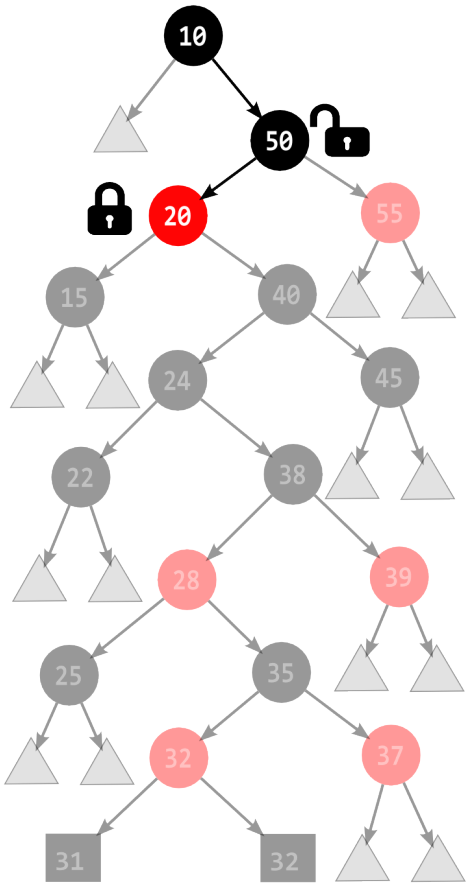
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

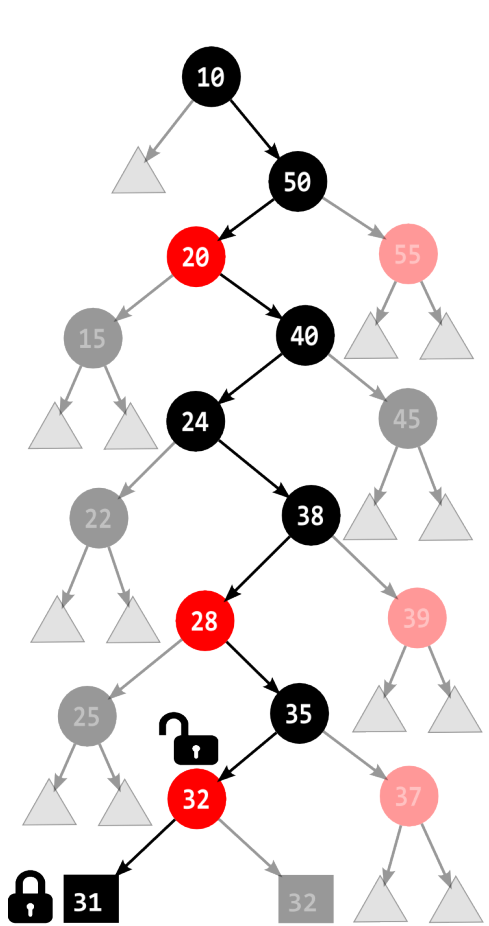
Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Example: lookup(31)



Red-Black tree operations: Lookup

Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

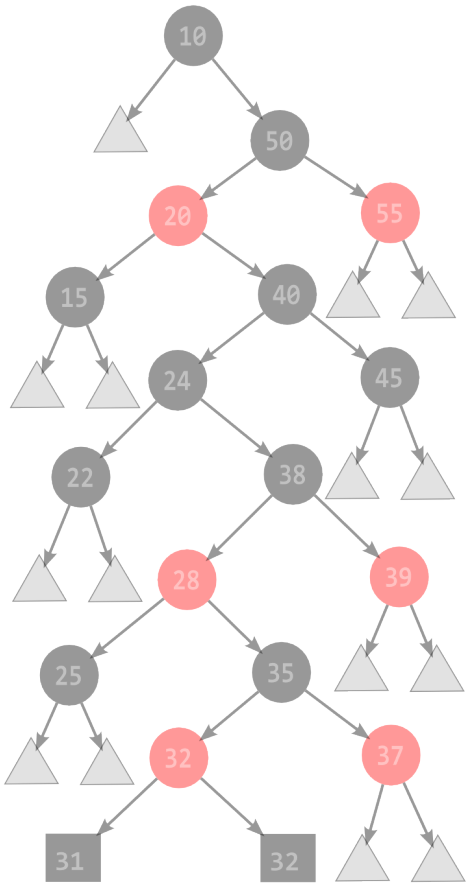
Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Coarse-grained HTM:

- Read-only transaction.
- Transactional read-set consists of the traversed nodes.

Example: lookup(31)



Red-Black tree operations: Lookup

Serial:

- Just like in BST.
- Traverse the tree from the root to the appropriate leaf.
- Lookups always reach a leaf (external tree).

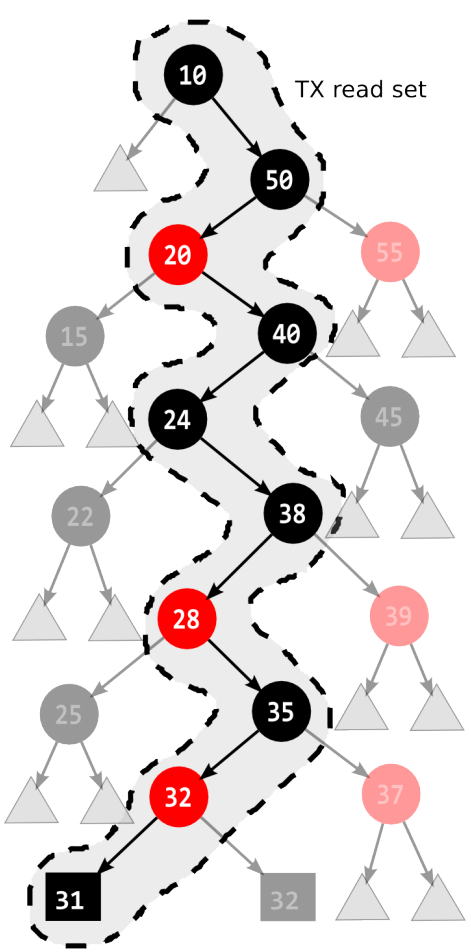
Fine-grained locking:

- Hand-over-hand locking.
- Lock next node before releasing current.

Coarse-grained HTM:

- Read-only transaction.
- Transactional read-set consists of the traversed nodes.

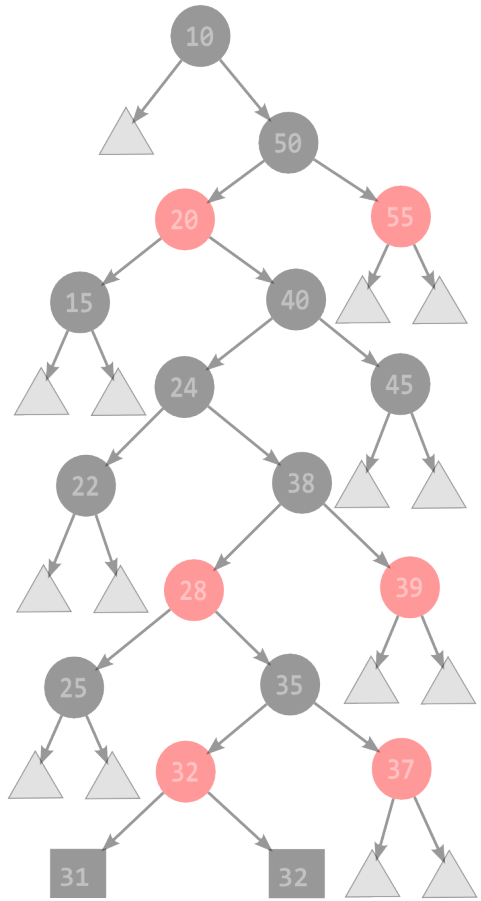
Example: lookup(31)



Red-Black tree operations: Insertion

Bottom-up [1]:

Example: insert(30)



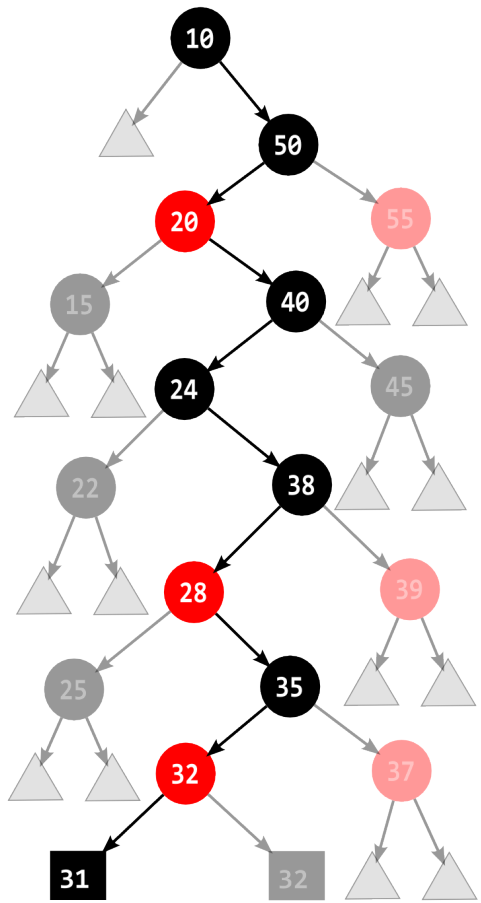
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.

Example: insert(30)



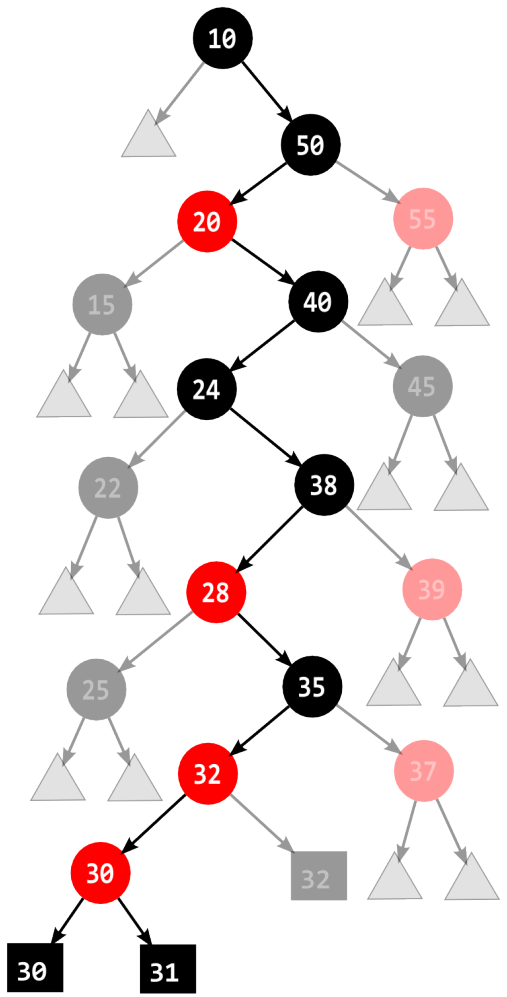
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.

Example: insert(30)



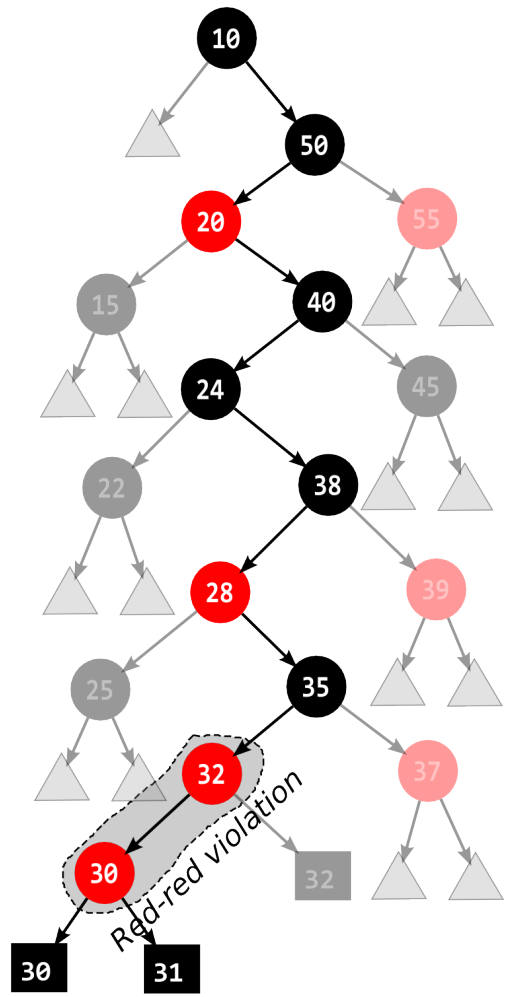
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Example: insert(30)



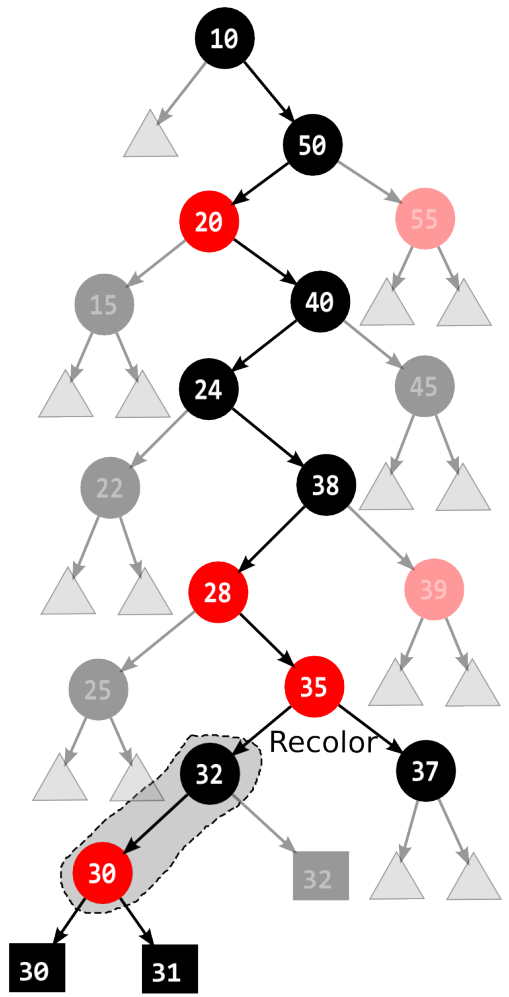
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Example: insert(30)



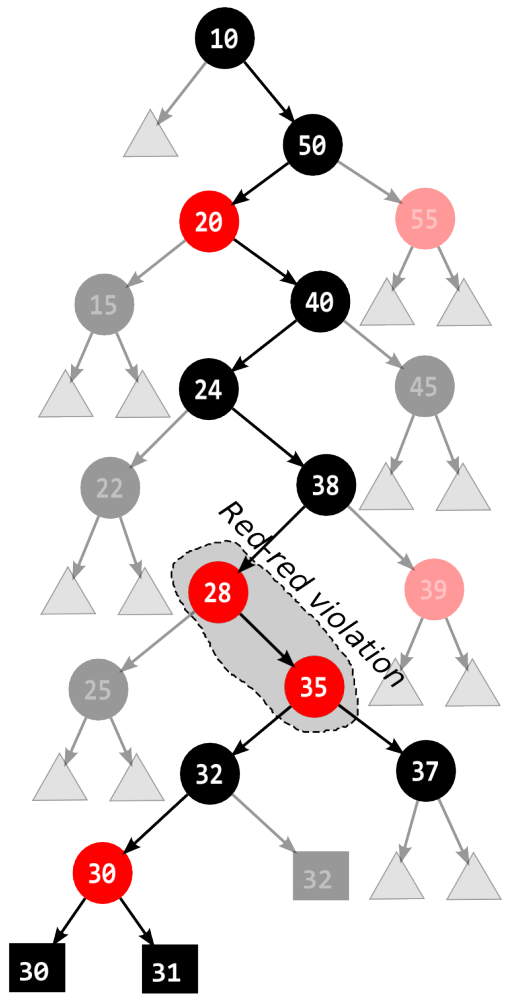
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Example: insert(30)



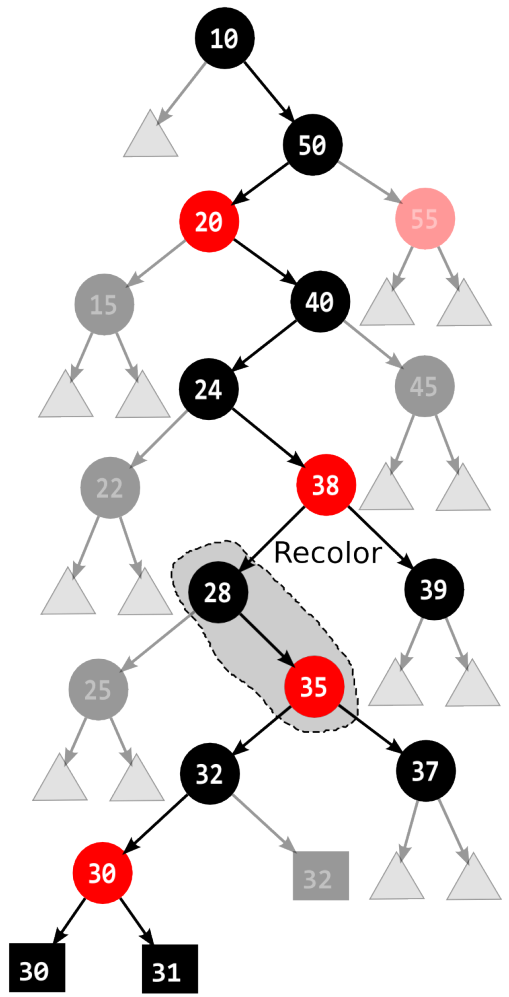
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Example: insert(30)



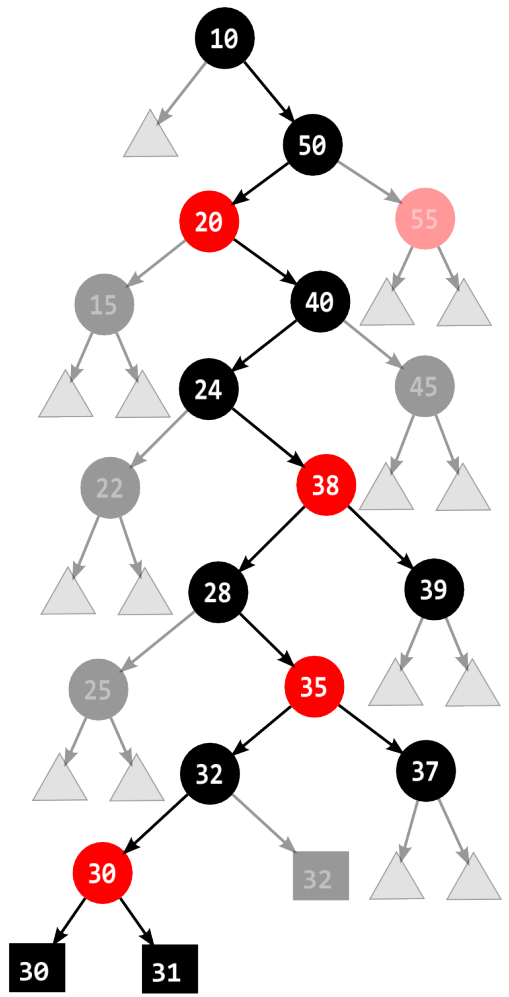
[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Example: insert(30)



[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

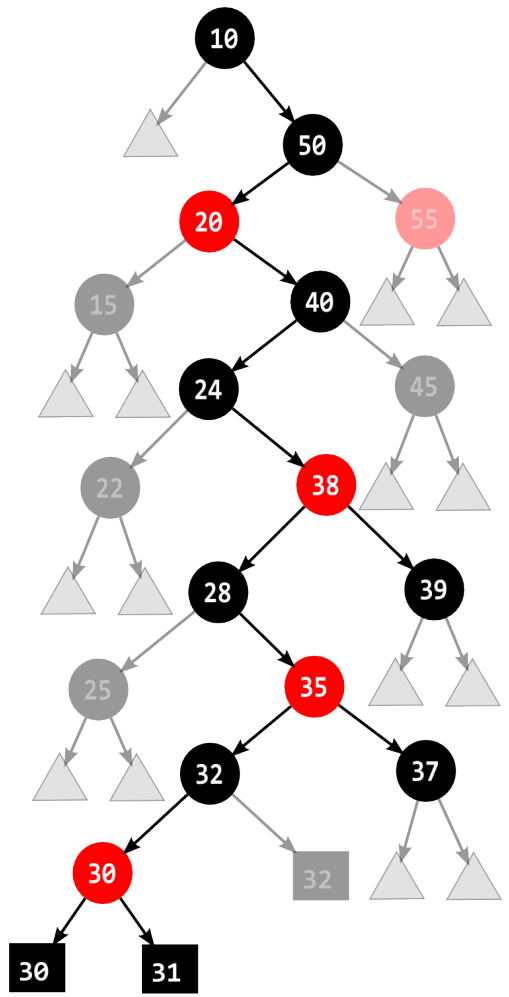
Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Fine-grained locking:

- Threads might traverse the tree in opposite directions.
- Deadlock.
- Top-down approach performs insertion in one pass.

Example: insert(30)



[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Fine-grained locking:

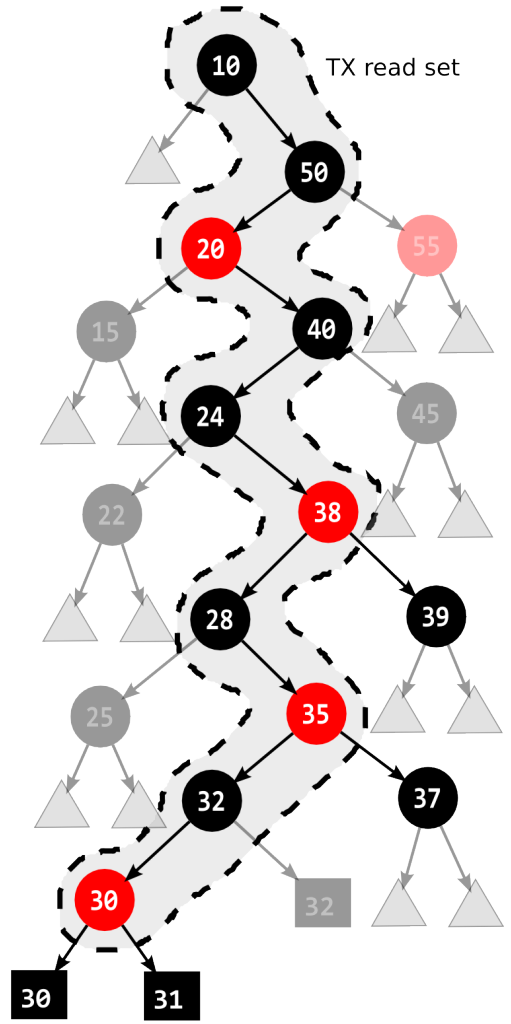
- Threads might traverse the tree in opposite directions.
- Deadlock.
- Top-down approach performs insertion in one pass.

Coarse-grained HTM:

- Transactional read-set consists of the traversed nodes plus nodes accessed in fixup phase.
- Transactional write-set consists of the nodes modified in fixup phase.

[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Example: insert(30)



Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Fine-grained locking:

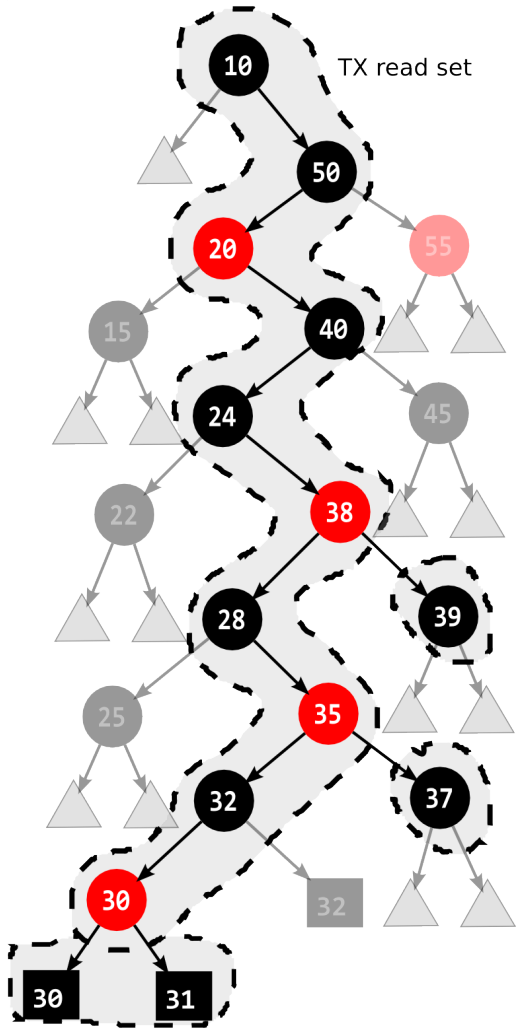
- Threads might traverse the tree in opposite directions.
- Deadlock.
- Top-down approach performs insertion in one pass.

Coarse-grained HTM:

- Transactional read-set consists of the traversed nodes plus nodes accessed in fixup phase.
- Transactional write-set consists of the nodes modified in fixup phase.

[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

Example: insert(30)



Red-Black tree operations: Insertion

Bottom-up [1]:

- Traverse the tree from the root to the appropriate leaf.
- Insert the new node.
- Fixup the violation (recolors and/or rotations).

Fine-grained locking:

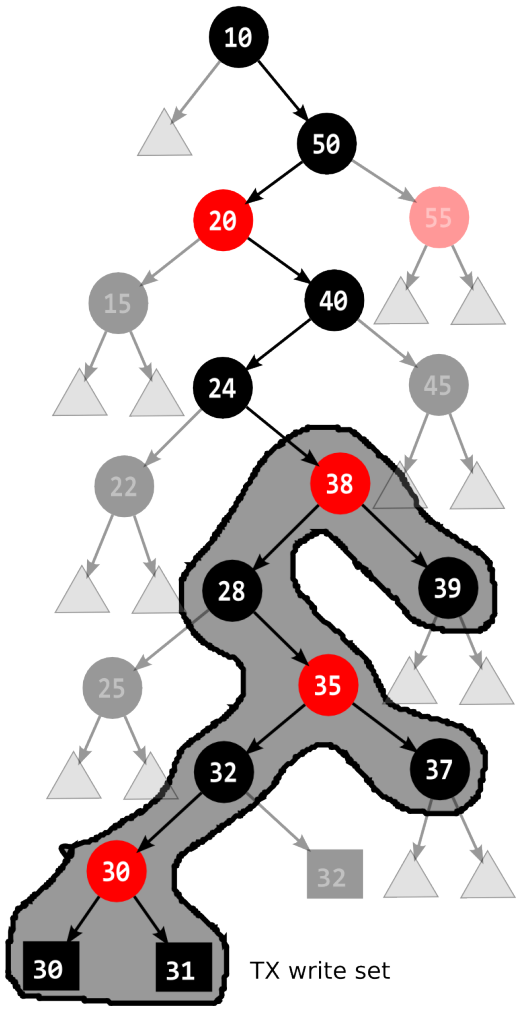
- Threads might traverse the tree in opposite directions.
- Deadlock.
- Top-down approach performs insertion in one pass.

Coarse-grained HTM:

- Transactional read-set consists of the traversed nodes plus nodes accessed in fixup phase.
- Transactional write-set consists of the nodes modified in fixup phase.

[1] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 3rd ed., 2009.

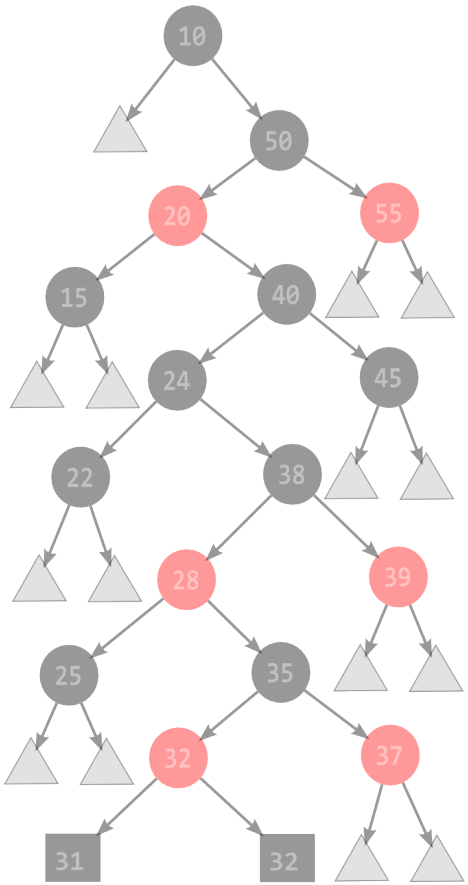
Example: insert(30)



Red-Black tree operations: Insertion

Top-down [1]:

Example: insert(30)



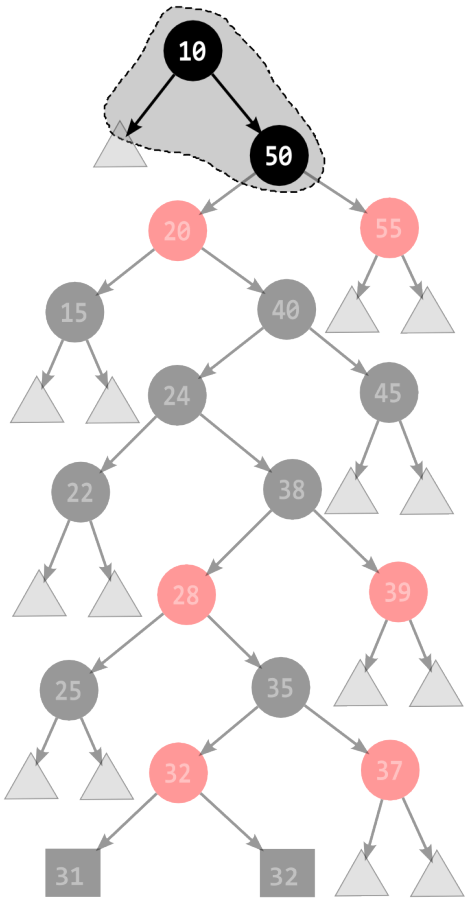
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



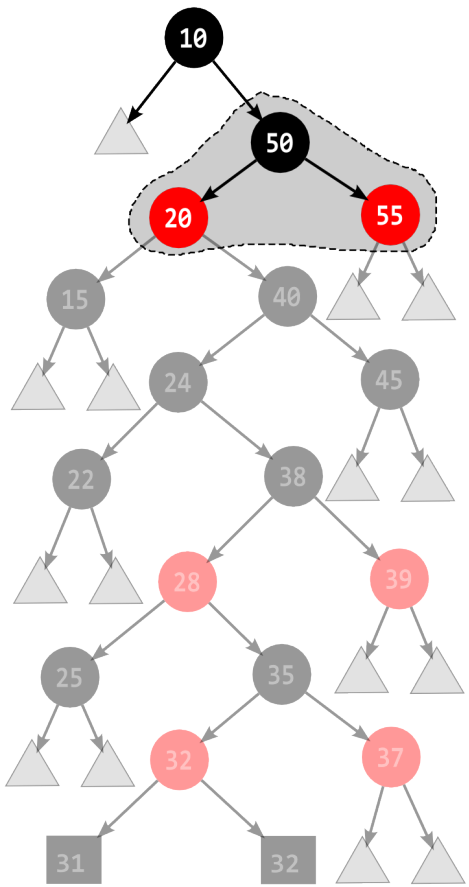
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



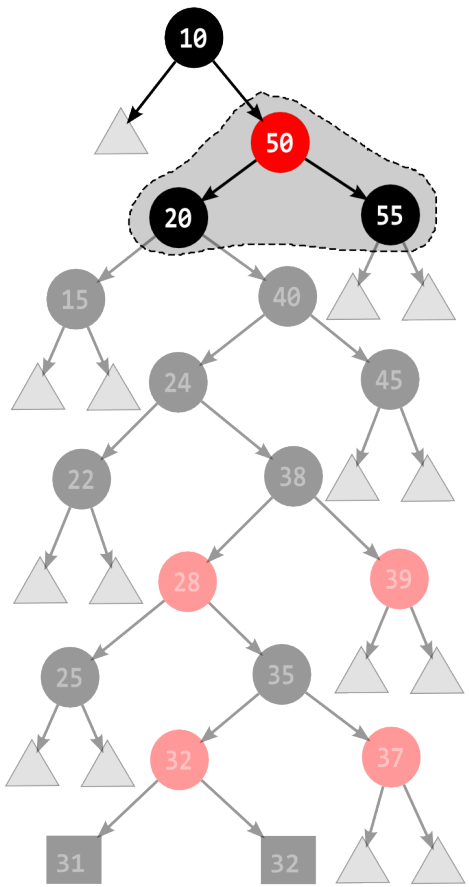
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



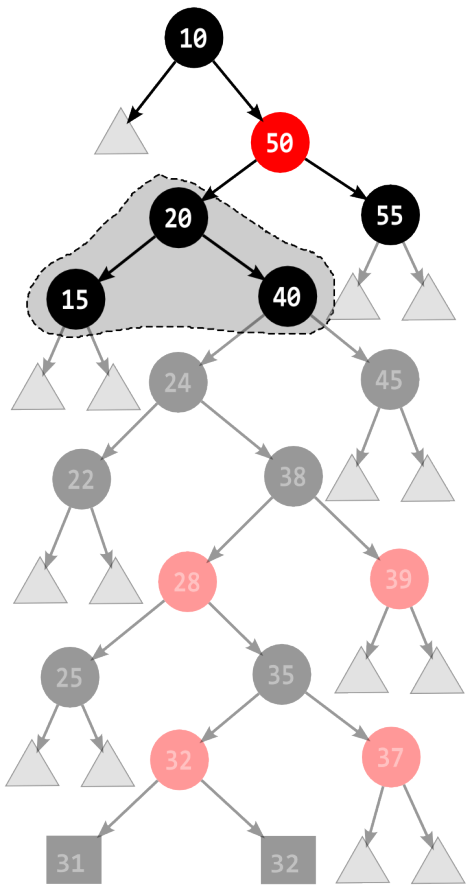
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



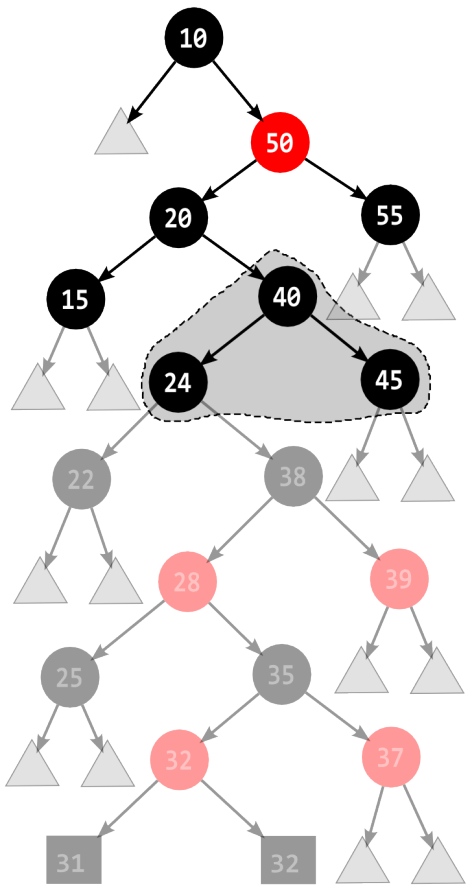
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



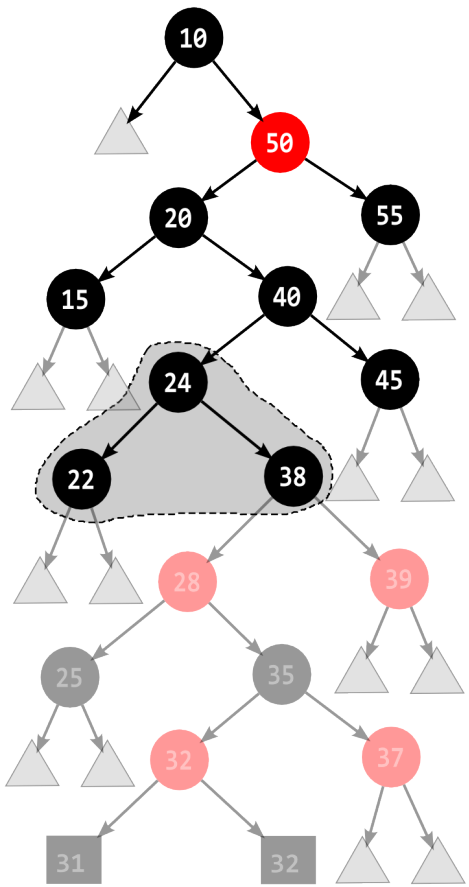
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



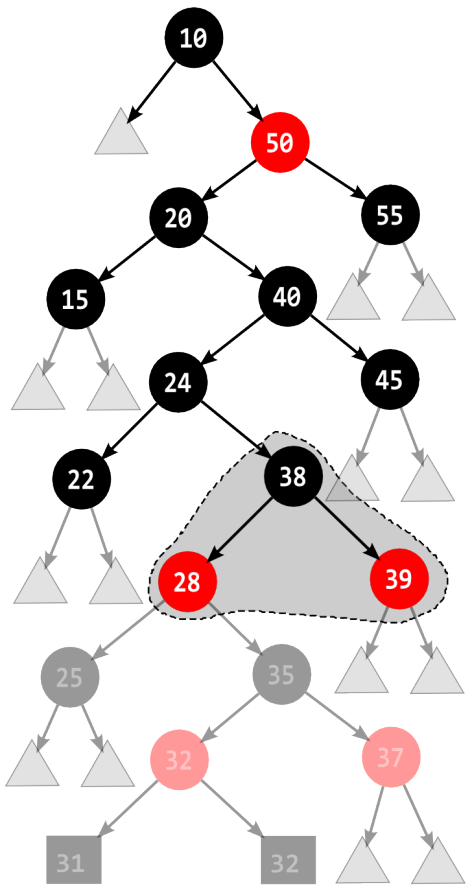
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



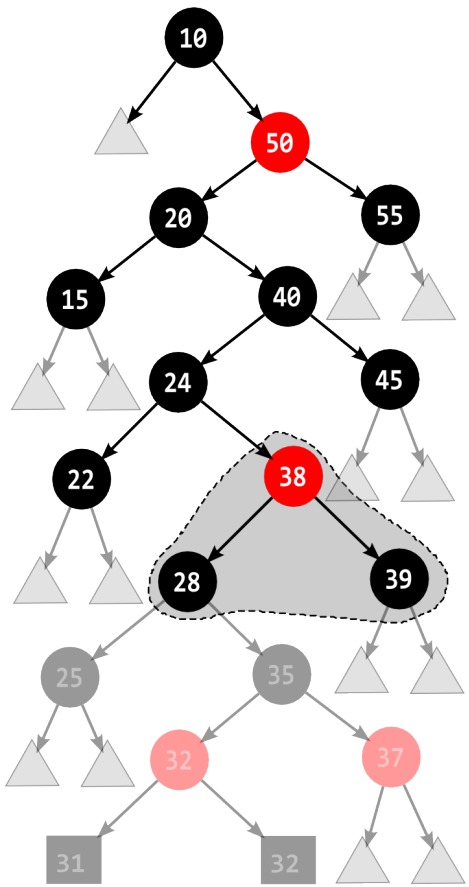
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



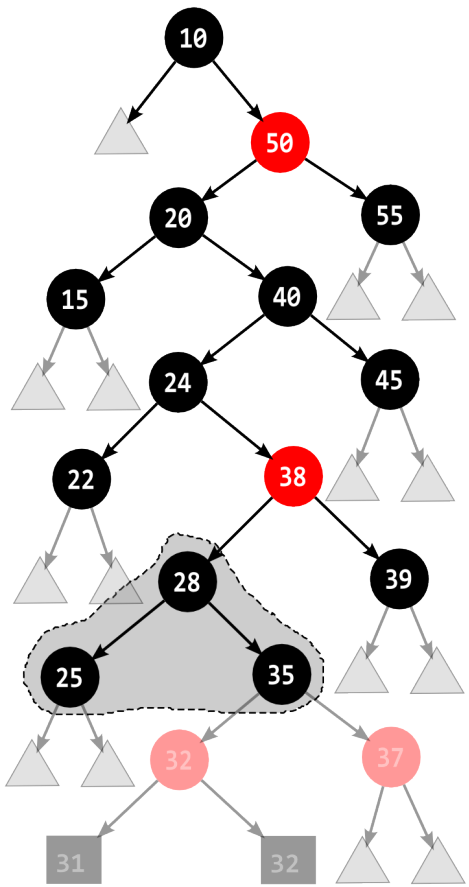
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



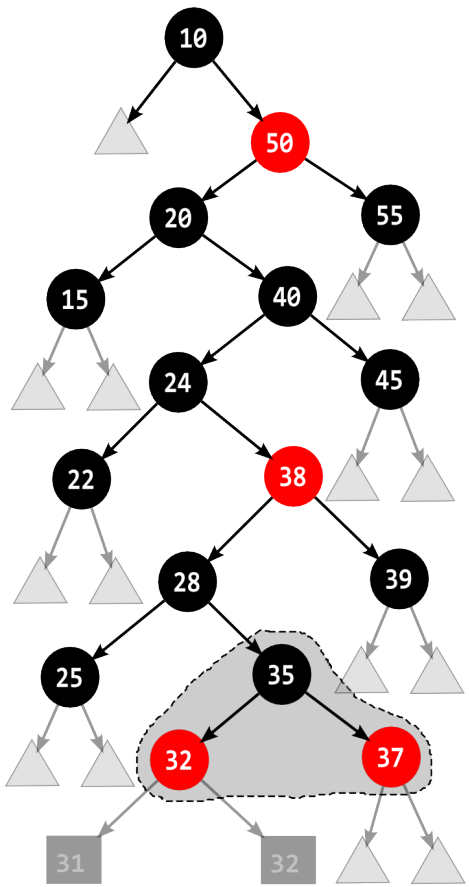
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



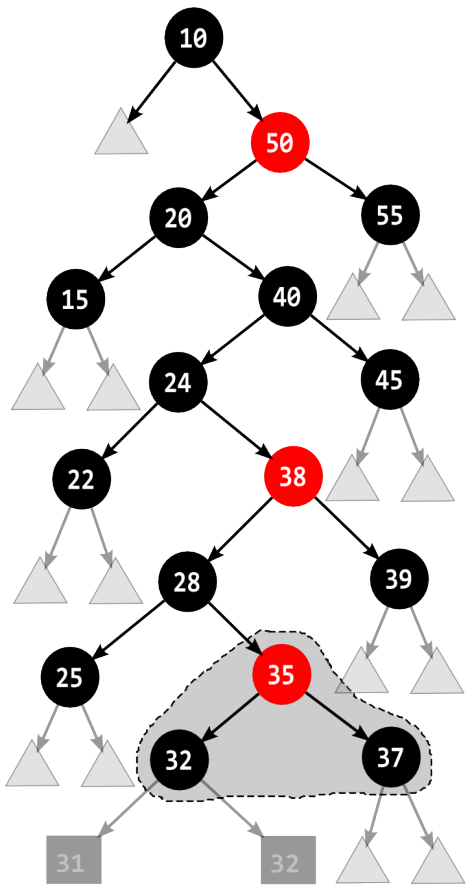
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



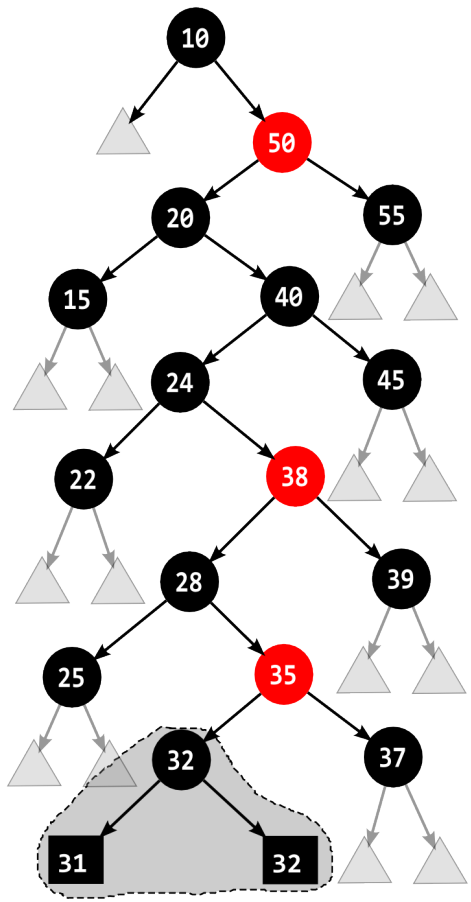
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.

Example: insert(30)



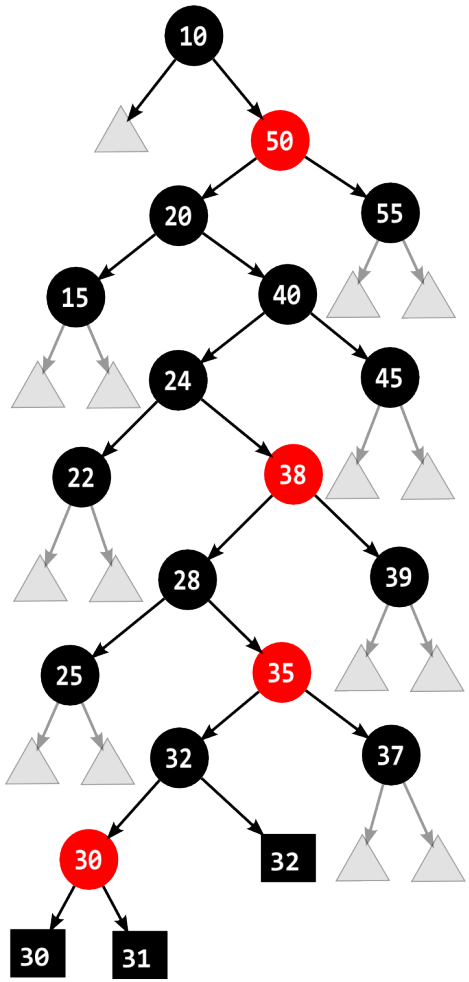
[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

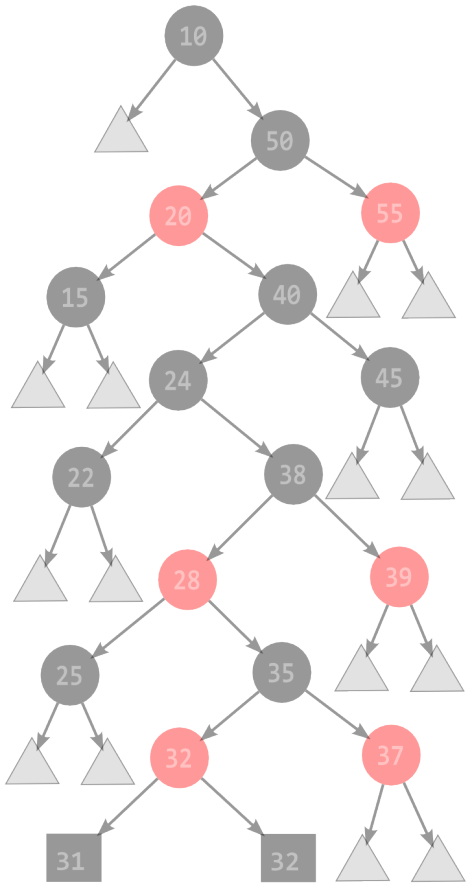
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

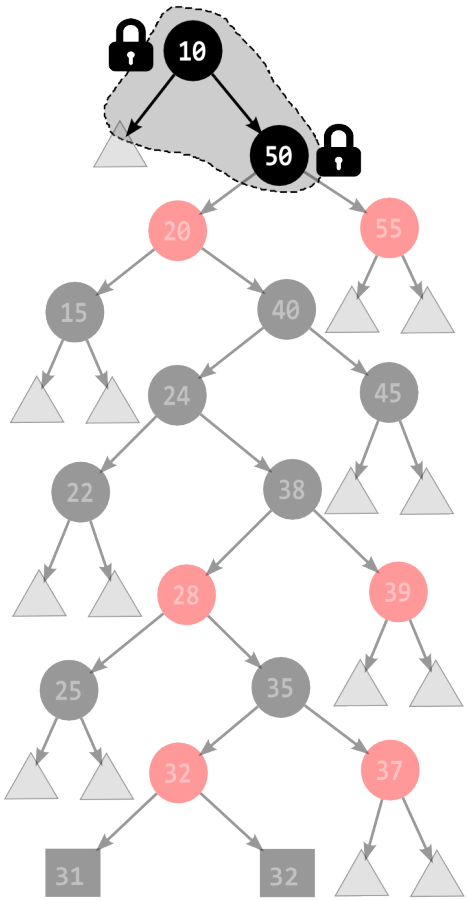
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

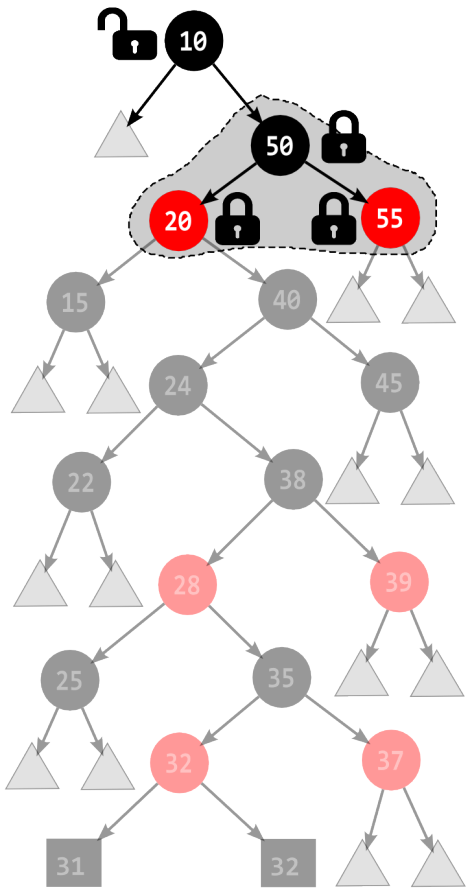
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

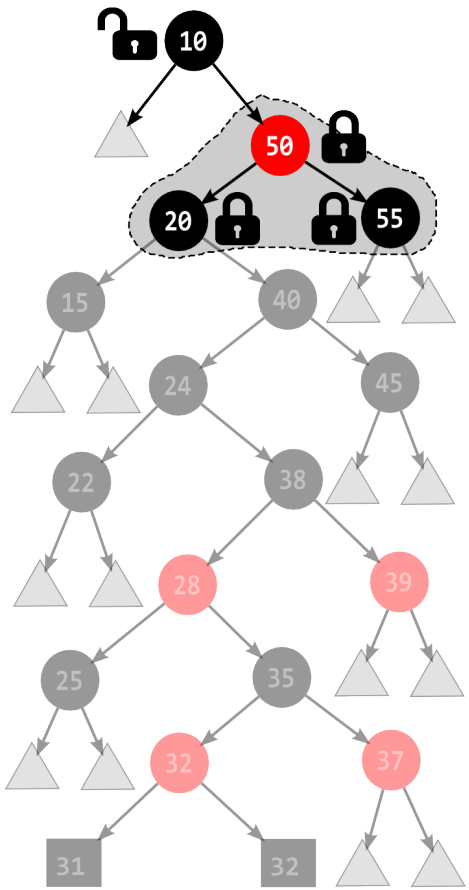
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

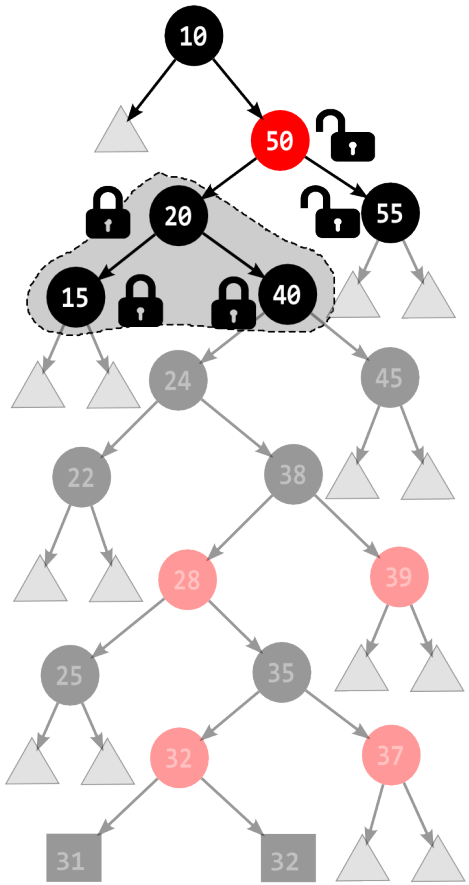
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

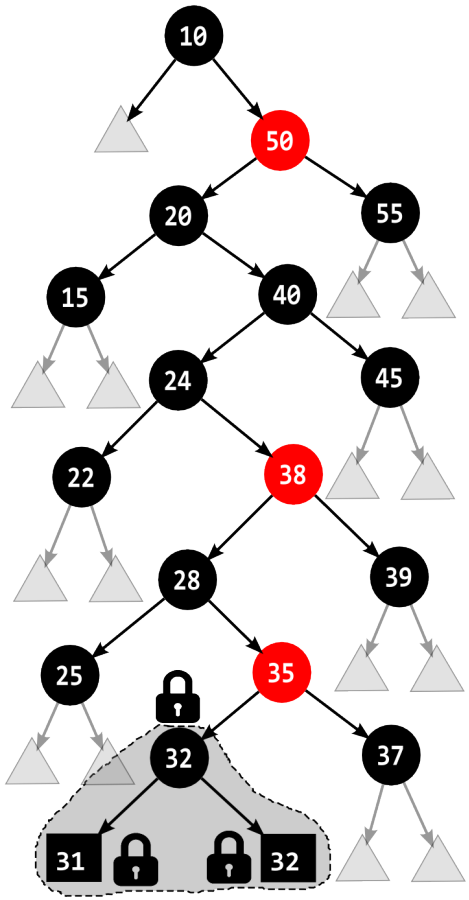
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

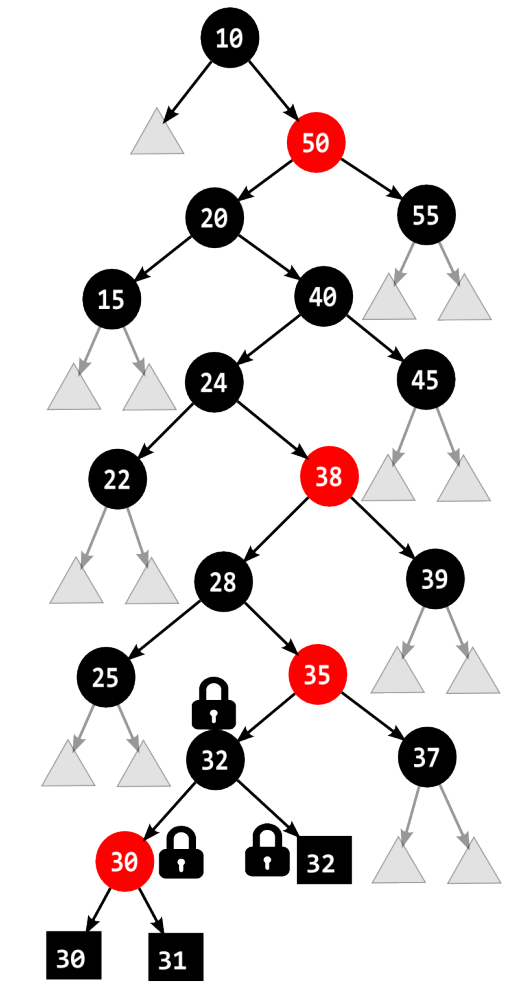
Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

- All threads traverse the tree in one direction.
- Avoids deadlock.

Example: insert(30)



[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

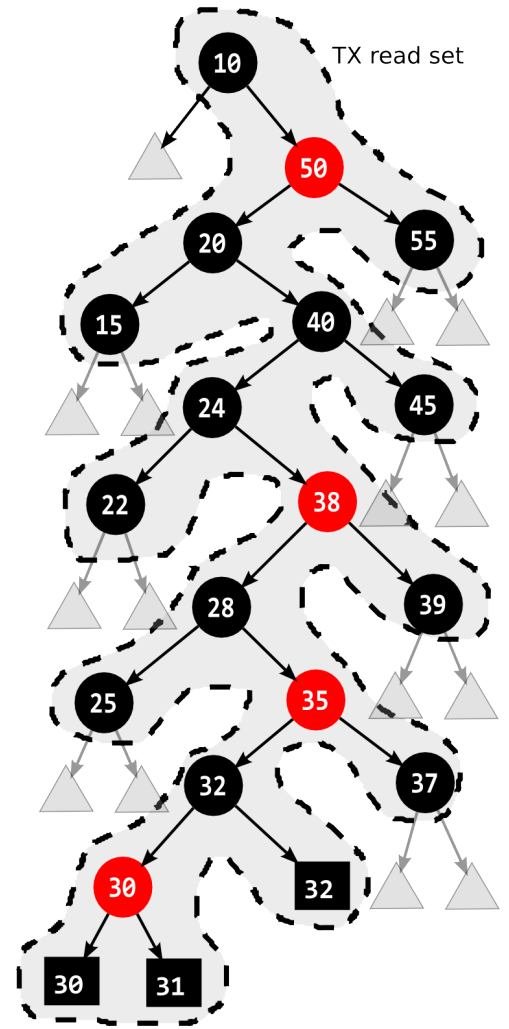
- All threads traverse the tree in one direction.
- Avoids deadlock.

Coarse-grained HTM:

- Transactional read-set consists of the nodes in the path plus their siblings.
- Transactional write-set consists of the nodes modified during traversal.

[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Example: insert(30)



Red-Black tree operations: Insertion

Top-down [1]:

- Traverse the tree recoloring/rotating nodes.
- Insert the new node.

Fine-grained locking:

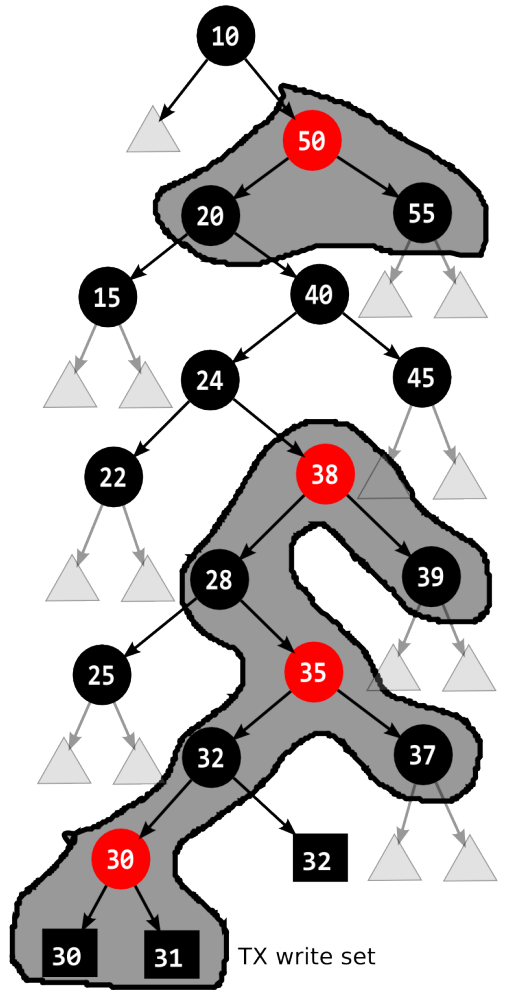
- All threads traverse the tree in one direction.
- Avoids deadlock.

Coarse-grained HTM:

- Transactional read-set consists of the nodes in the path plus their siblings.
- Transactional write-set consists of the nodes modified during traversal.

[1] R. A. Tarjan, *Efficient top-down updating of red-black trees*, Tech. Rep. TR-006-85, Department of Computer Science, Princeton University, 1985.

Example: insert(30)



Red-Black tree operations: Deletion

Bottom-up:

- Traverse the path to the appropriate node.
- Delete the node.
- Fixup the violation.
  - Similar to bottom-up insertion.
  - Recolors propagate the violation and, if necessary, 1 2 or 3 rotations

Top-down:

- One top-down pass.
- Recolors/rotations while traversing the tree to assure no reverse traversal is necessary.

Concurrent red-black tree implementations

	Bottom-up	**Top-down**
Coarse-grained lock	*bu_cg_lock*	-
Fine-grained lock	-	*td_fg_lock*
Coarse-grained HTM	*bu_cg_htm*	*td_cg_htm*

HTM

Current HTM platforms

<table>
<tr><th></th><th>Intel Haswell</th><th>IBM Power8</th></tr>
<tr><td>Versioning</td><td colspan="2">Lazy</td></tr>
<tr><td>Progress guarantees</td><td colspan="2">Best effort</td></tr>
<tr><td>Conflict detection</td><td colspan="2">Eager</td></tr>
<tr><td>Conflict granularity</td><td colspan="2">Cache line</td></tr>
<tr><td>Cache line size</td><td>64KB</td><td>128KB</td></tr>
<tr><td>TX read-set</td><td>4MB</td><td>8KB</td></tr>
<tr><td>TX write-set</td><td>22KB</td><td>8KB</td></tr>
<tr><td rowspan="4">Failure reasons</td><td rowspan="2">Conflict</td><td>Transactional conflict</td></tr>
<tr><td>Non-transactional conflict</td></tr>
<tr><td colspan="2">Capacity</td></tr>
<tr><td colspan="2">Explicit</td></tr>
</table>

Experimental Evaluation

Experimental Platforms

Name	**Haswell**	**Power8**
Processors	1 x Intel Core i7-4771	2 x Power8
# Cores	4	2 x 10
# Threads	8	160
Core clock	3.5 GHz	3.7 GHz
L1 (Data)	8-way, 32 KB, 64B block size	8-way, 64 KB, 128B block size
L2	8-way, 256 KB, 64B block size	8-way, 512 KB, 128B block size
L3	16-way, 8 MB, 64B block size (shared)	8-way, 80 MB, 128B block size (shared per die)
Memory	16 GB	256 GB

Experimental Setup

Code

- Written in C.
- GCC 4.9.1, -O3 optimization flag used.
- Nodes are padded to fit in one cache line.

HTM Evaluation

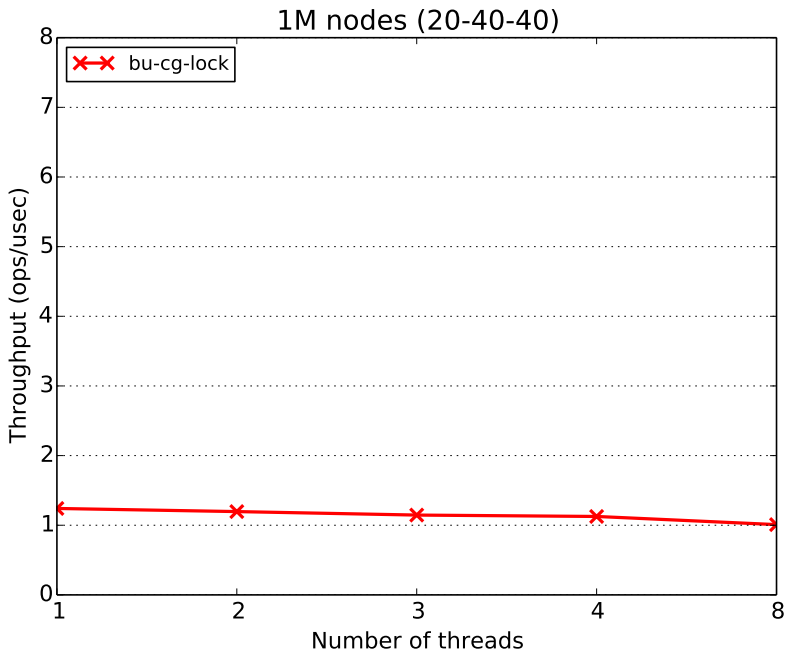
- Tree sizes (5 configurations):
  - Small: 1K nodes $\rightarrow$ 9-12 nodes in paths, $\approx$250KB.
  - Large: 10M nodes $\rightarrow$ 19-29 nodes in paths, $\approx$2.5GB.
- Operations workload (%lookups-%insertions-%deletions):
  - Read-intensive: 80-10-10
  - Read-write: 50-25-25
  - Write-intensive: 20-40-40

Benchmarks Execution

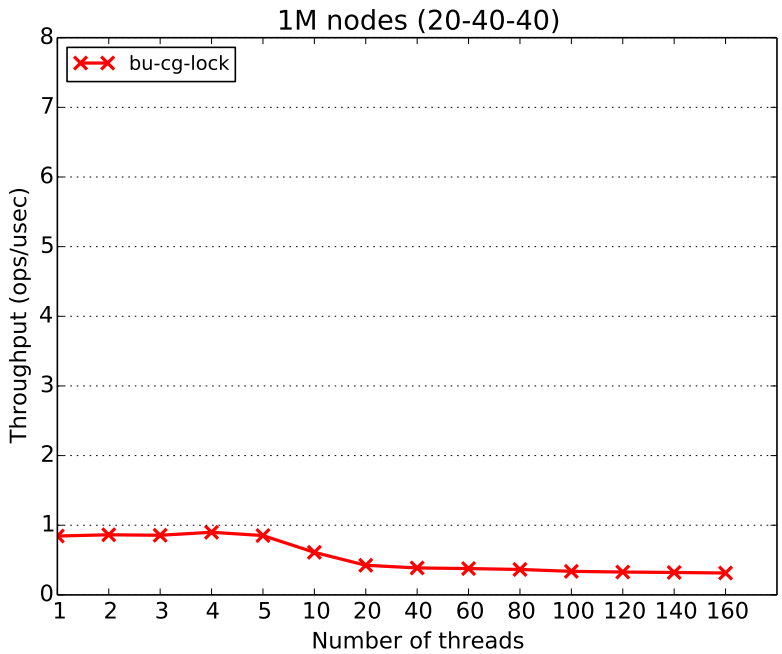
- Employ empty physical cores before SMT contexts.
- RTM mode only in Haswell.
- 10M operations equally divided between threads.
- Warmup phase to initialize the tree with half of the possible keys.
- 10 transactional retries before acquiring the global lock.

Evaluation: HTM VS Locks

Haswell

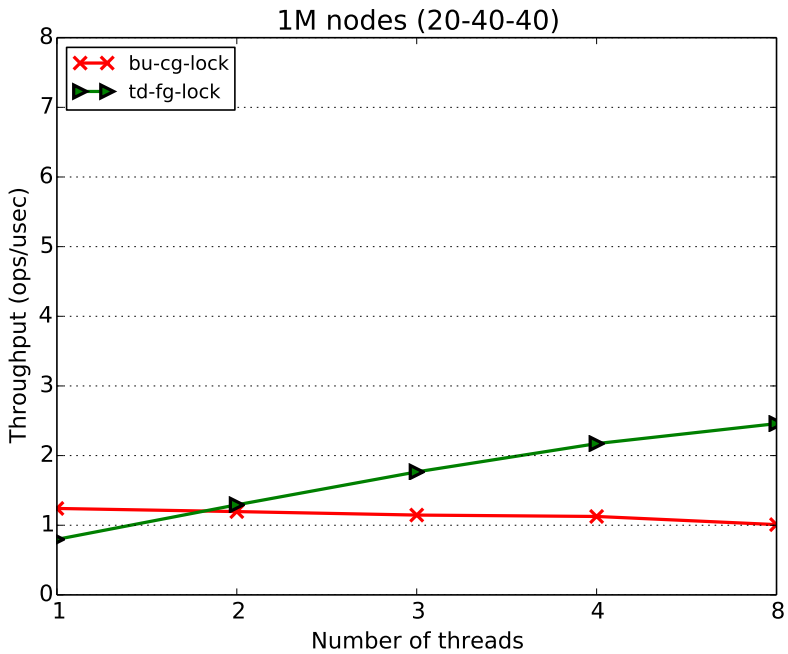


Power8

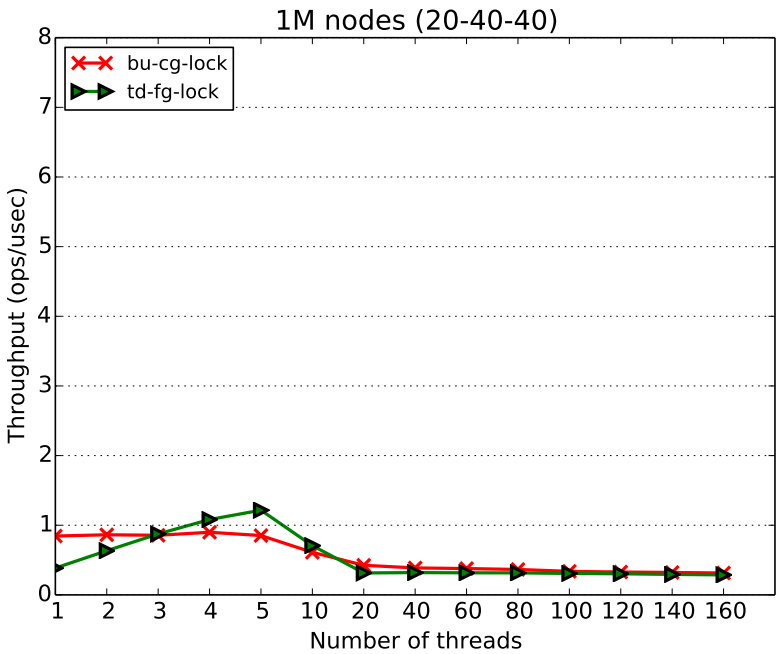


Evaluation: HTM VS Locks

Haswell

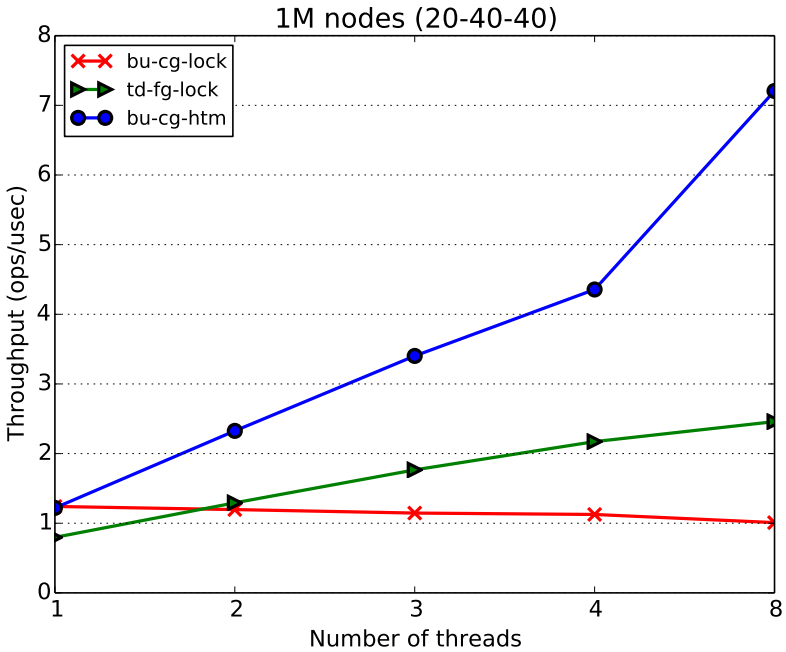


Power8

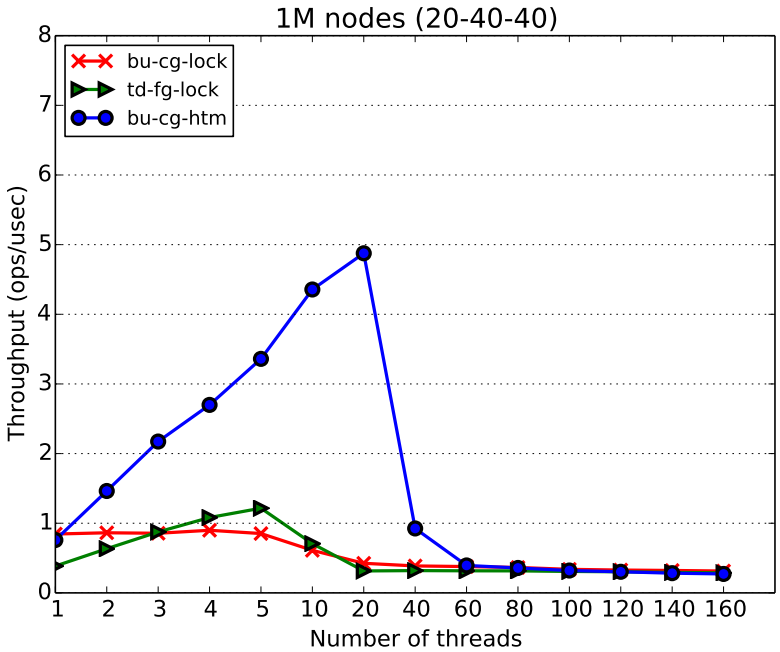


Evaluation: HTM VS Locks

Haswell

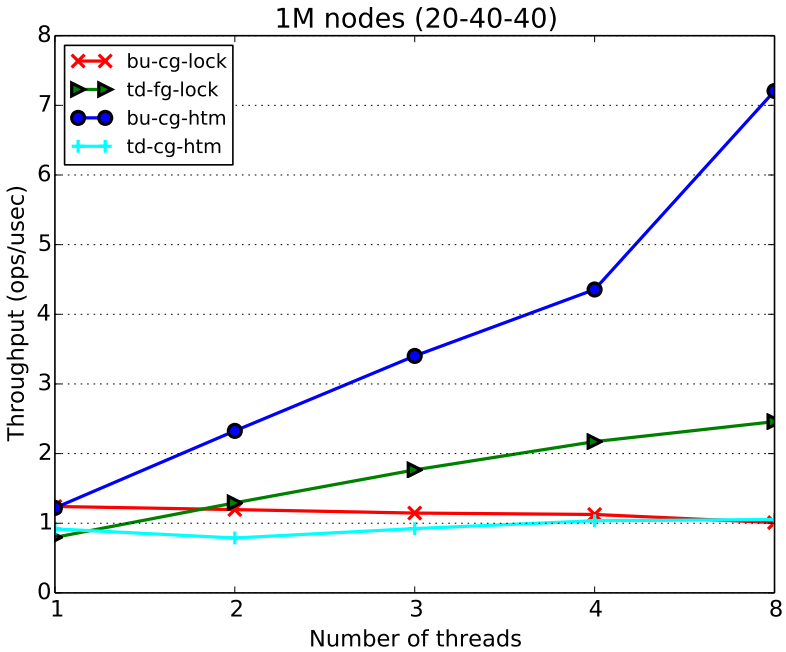


Power8

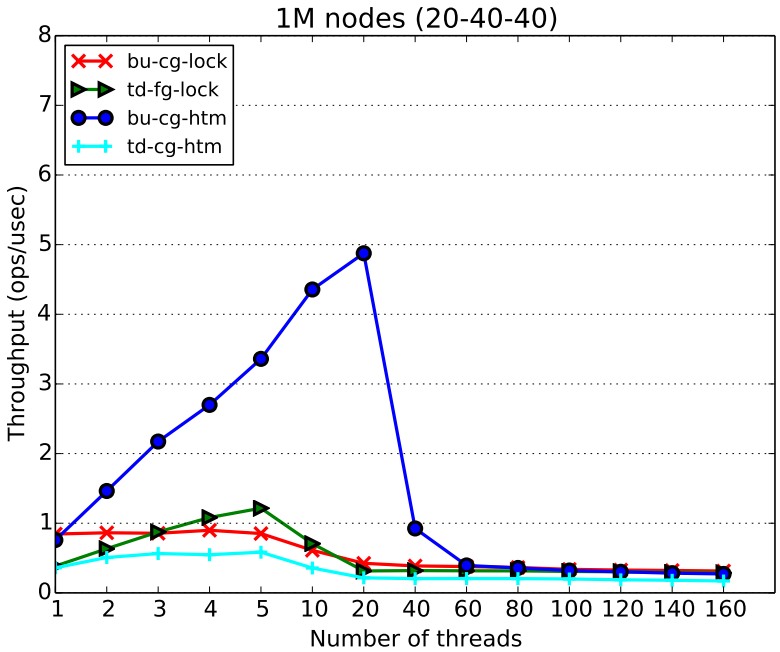


Evaluation: HTM VS Locks

Haswell

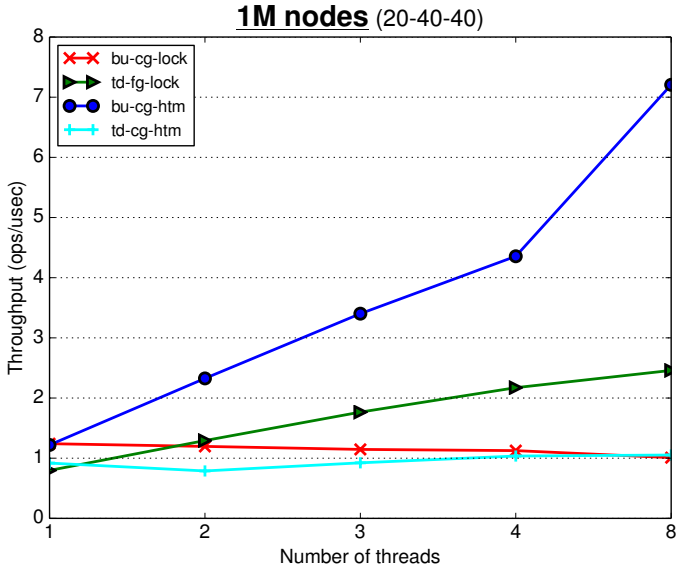


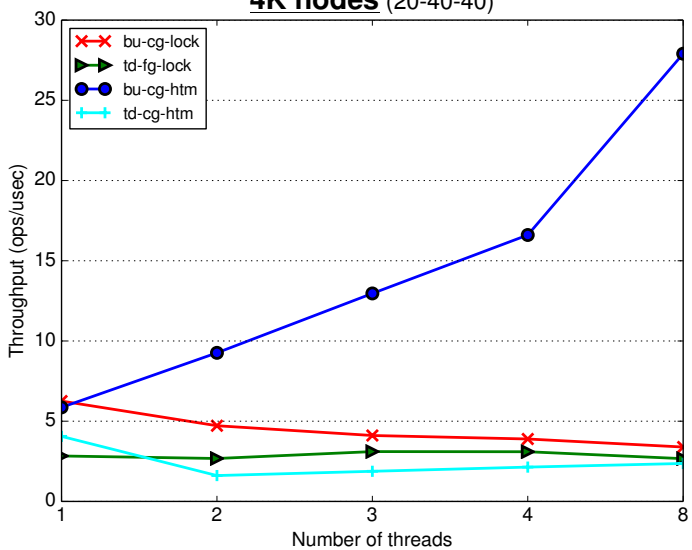
Power8



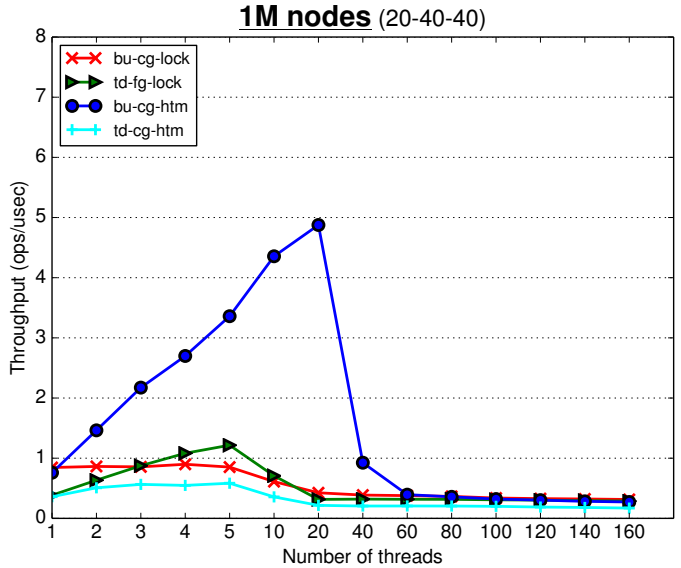
Evaluation: HTM VS Locks

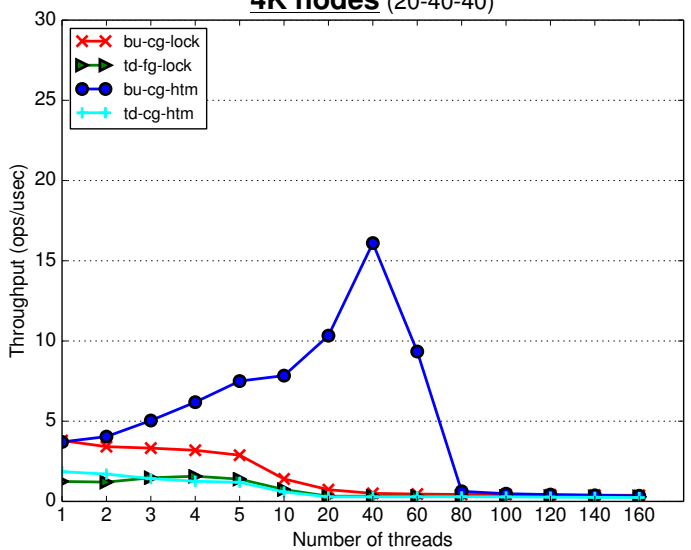
Haswell





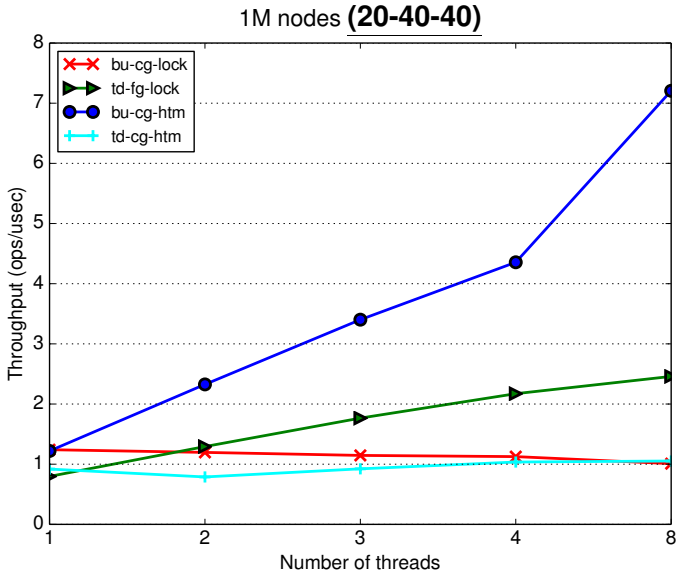
Power8

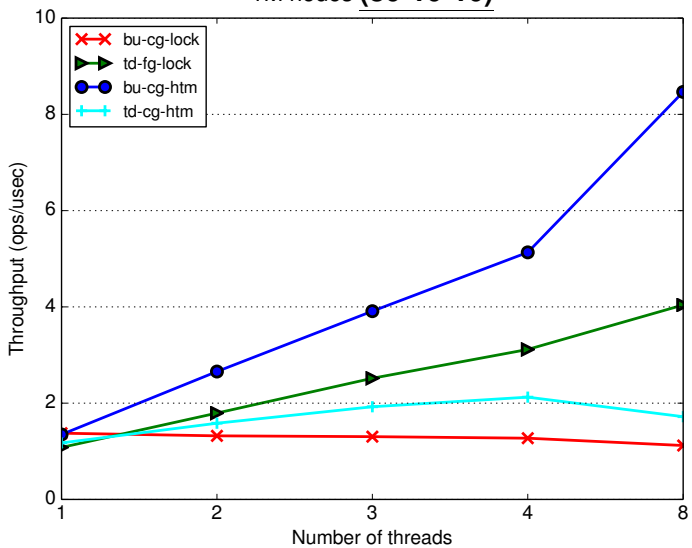




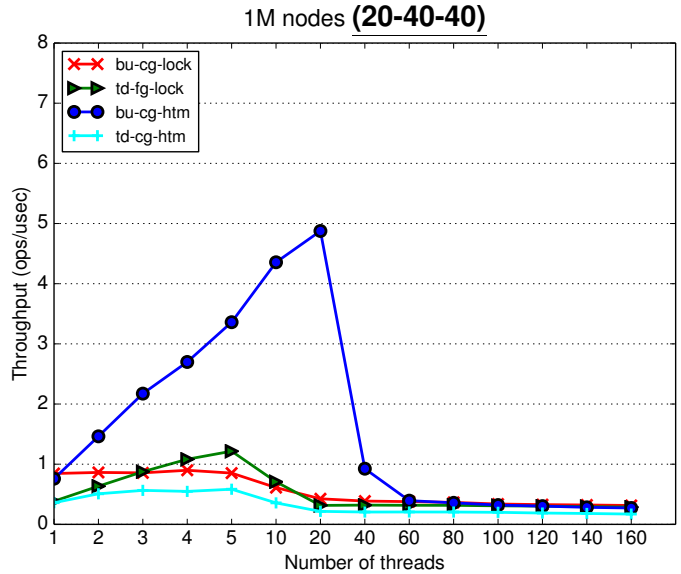
Evaluation: HTM VS Locks

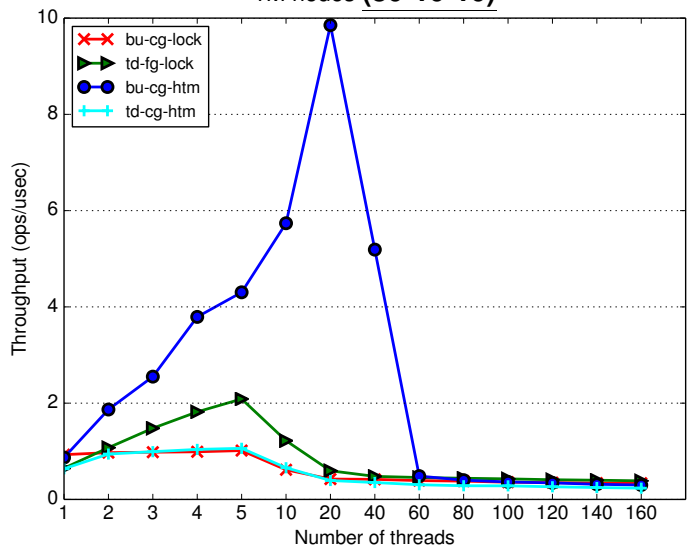
Haswell





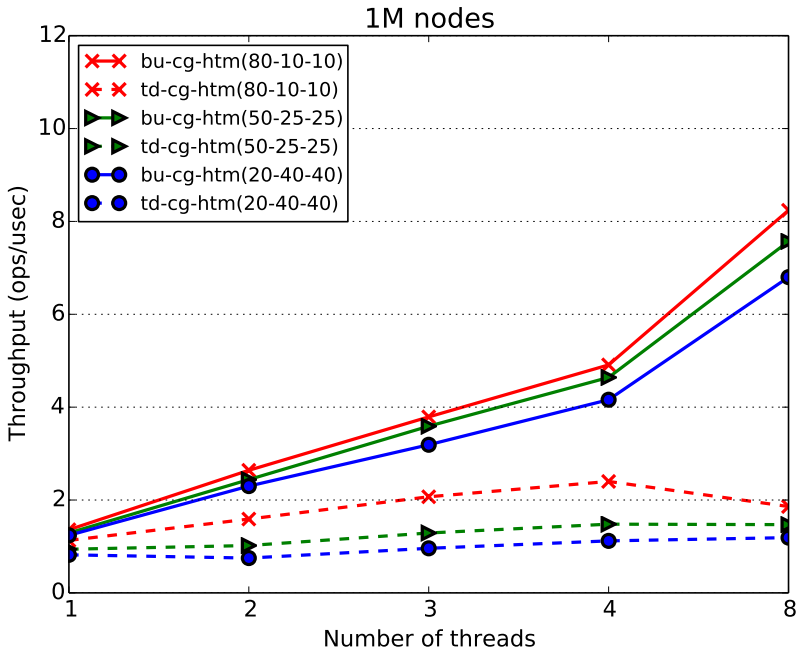
Power8



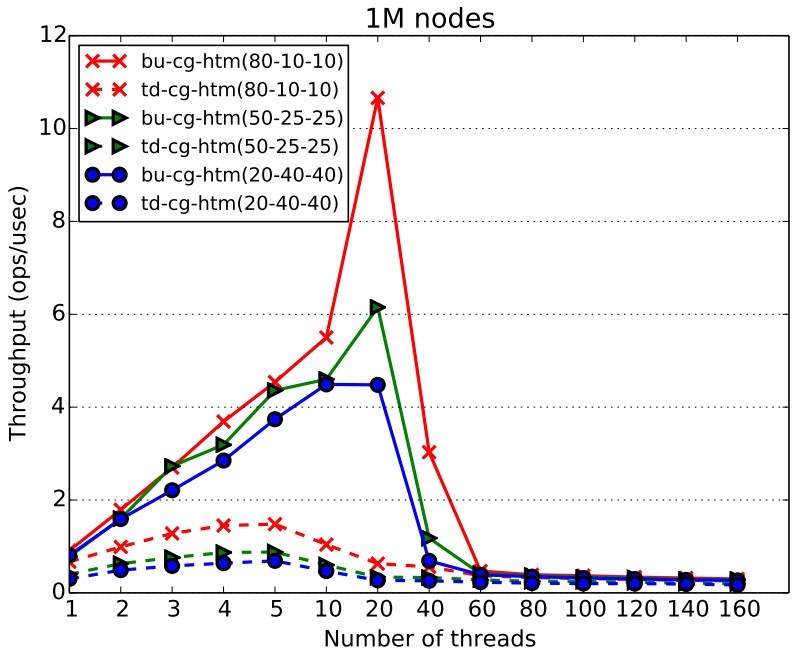


Evaluation: HTM Bottom-up VS Top-down

Haswell

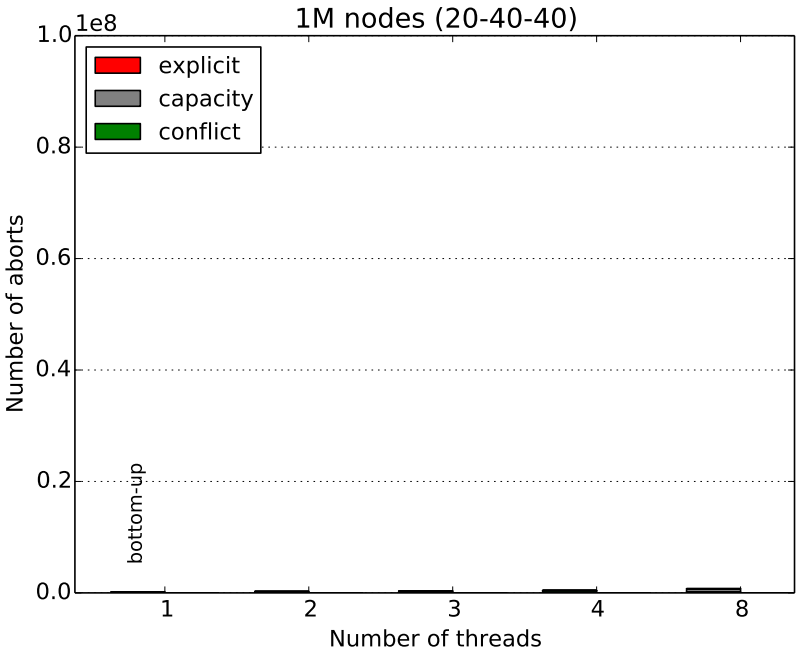


Power8

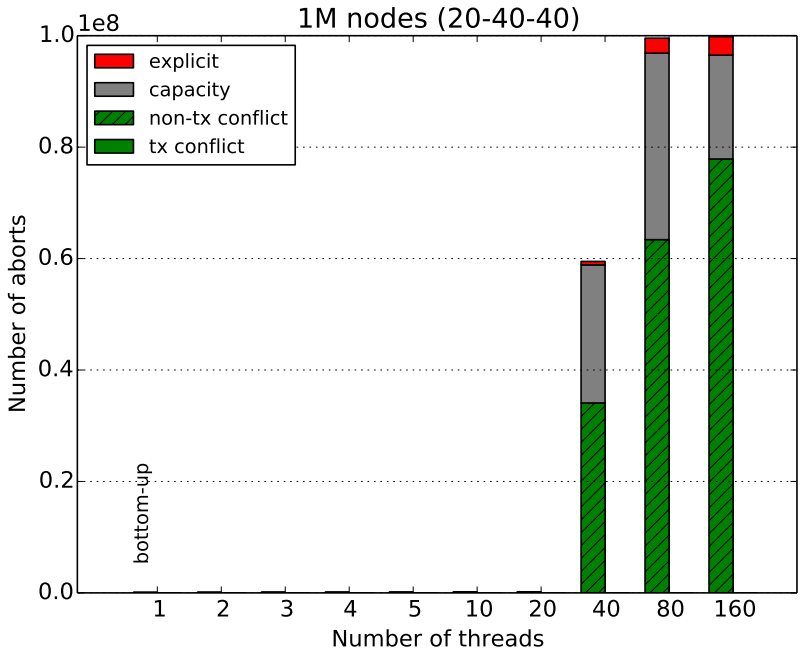


Evaluation: HTM Bottom-up VS Top-down

Haswell

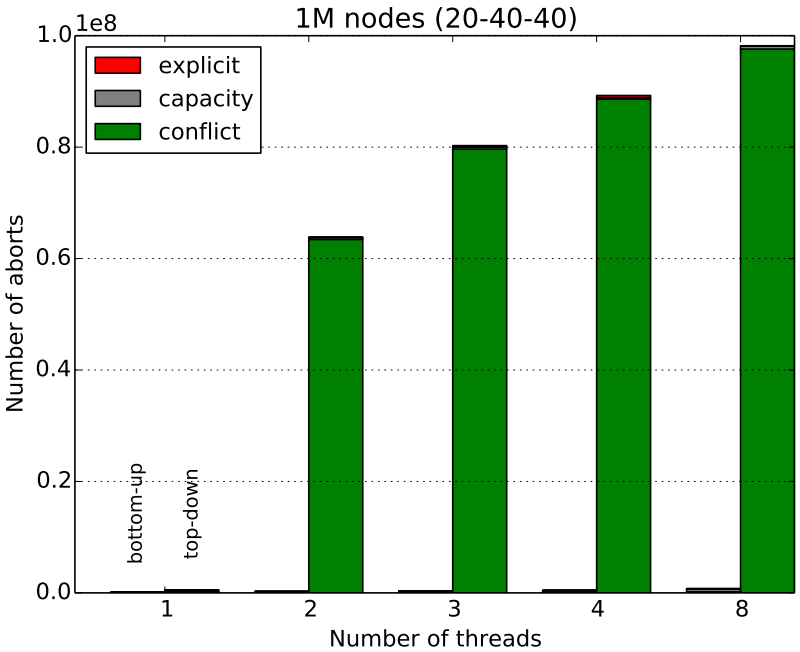


Power8

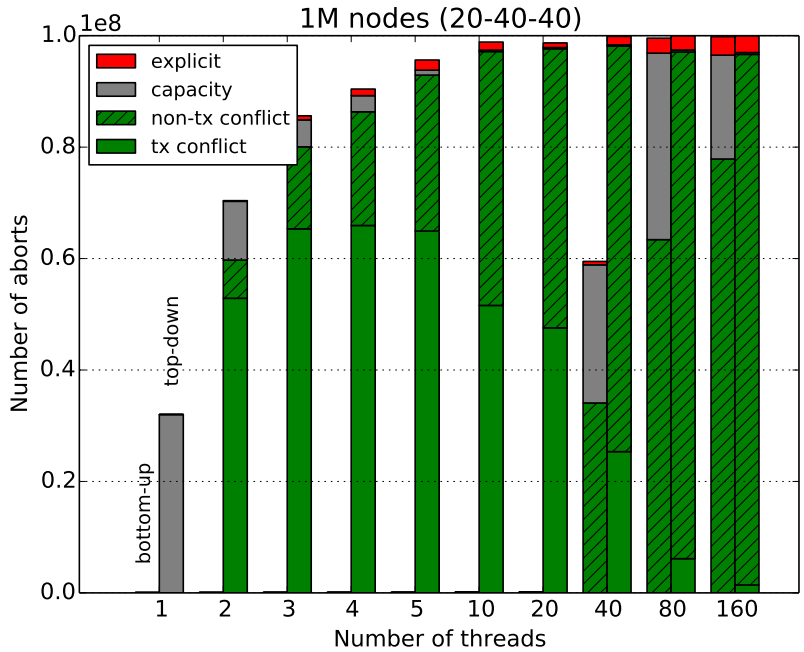


Evaluation: HTM Bottom-up VS Top-down

Haswell

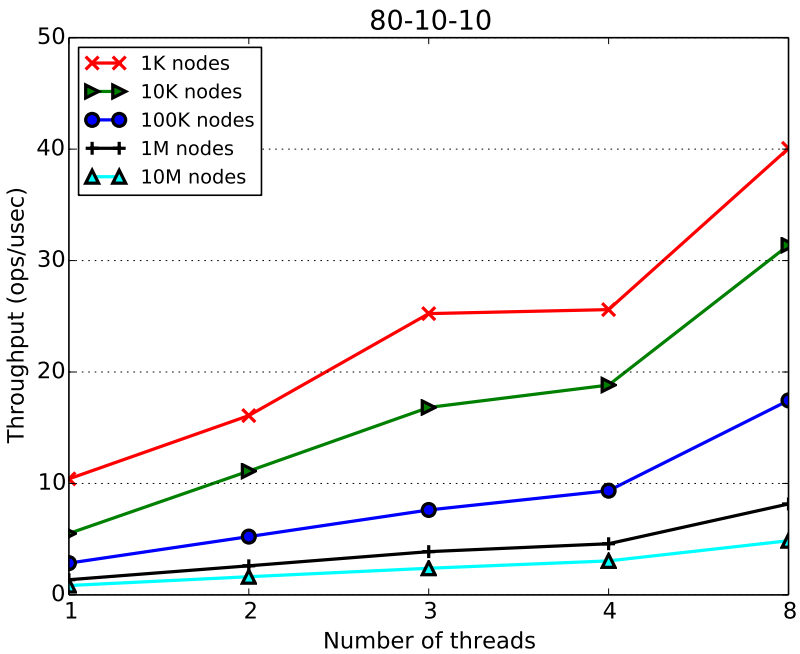


Power8

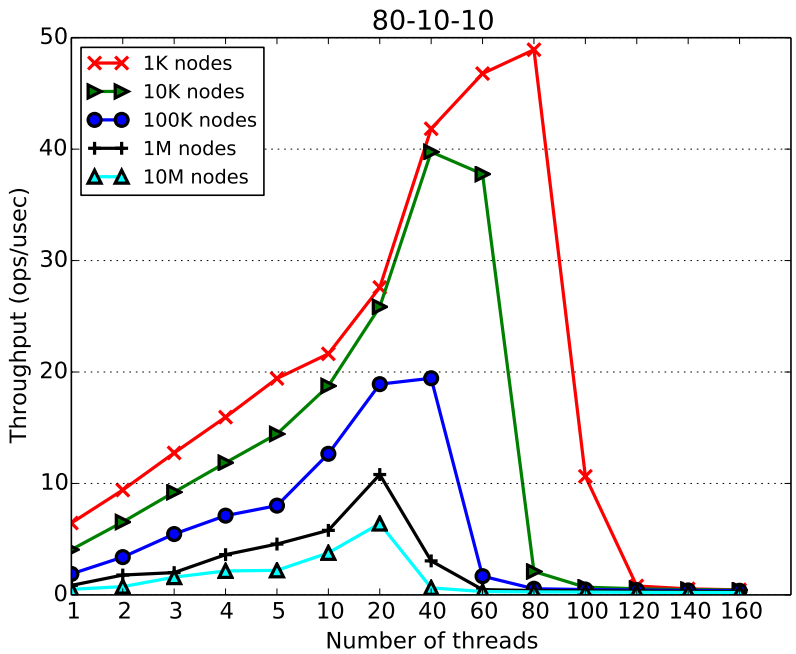


Evaluation: Effects of HTM imposed limits

Haswell



Power8



Conclusions

- HTM:
  - Better than fine-grained locking ...
  - ... with programming effort similar to coarse-grained locking.
- Hardware imposes performance limits.
  - More intense when resourses are shared.

Future Directions

- Cpu_lock fallback
- Hand-over-hand transactions
- More data structures, e.g. AVL trees, B+ trees ...
- ... any suggestions?

Preliminary results on my lightning talk :)

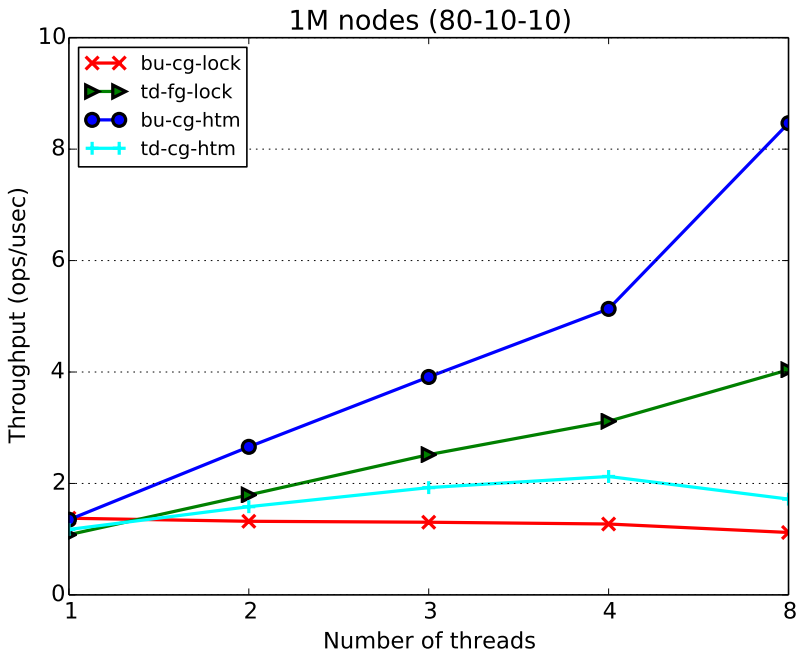
Thank you for your attention!

Questions?

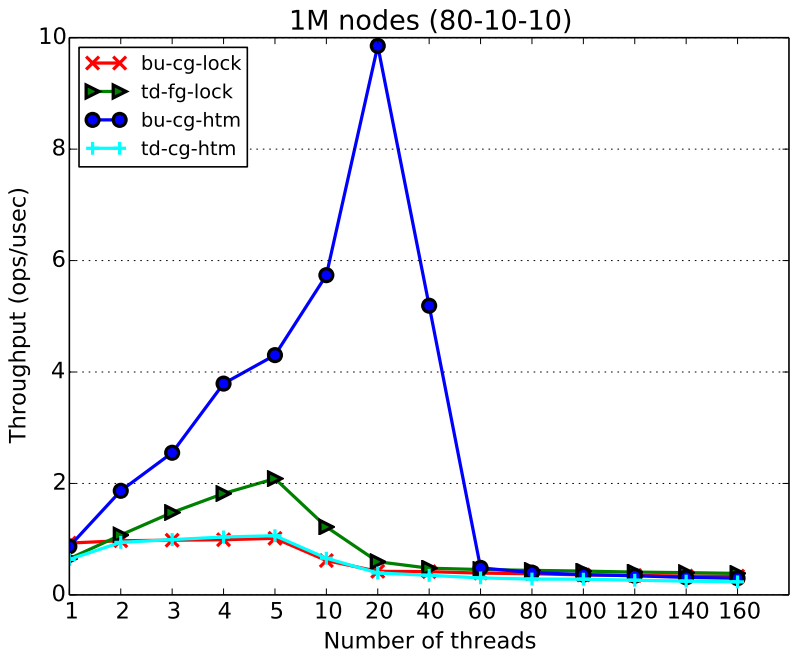
Backup

HTM VS Locks: 1M nodes (80-10-10)

Haswell

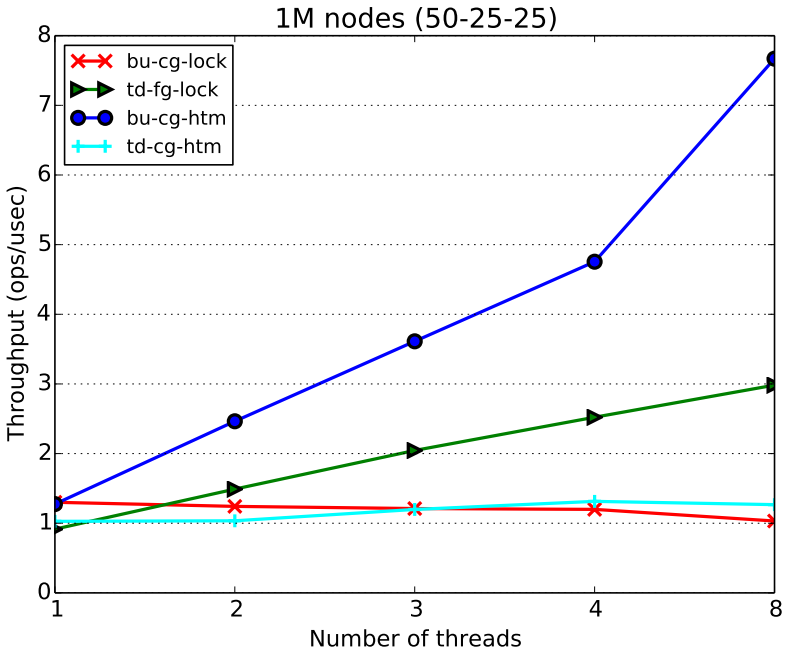


Power8

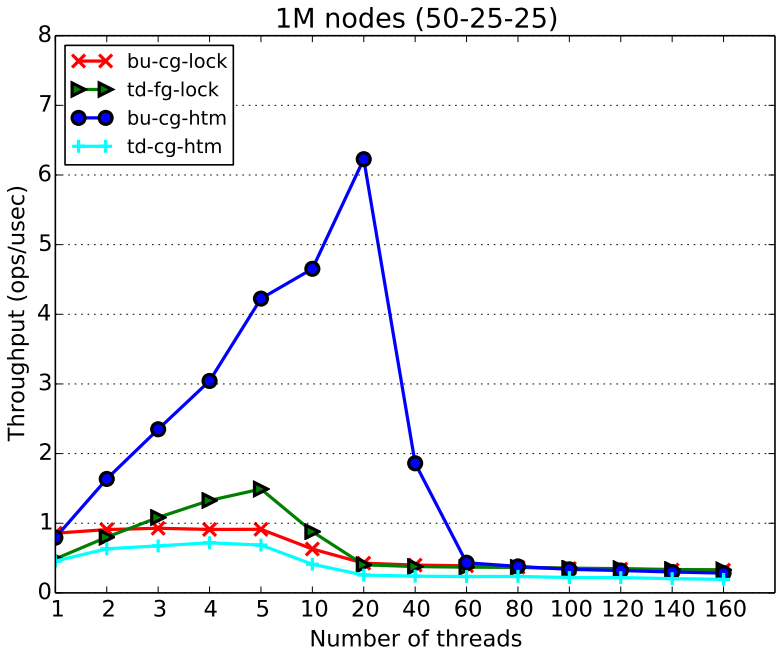


HTM VS Locks: 1M nodes (50-25-25)

Haswell

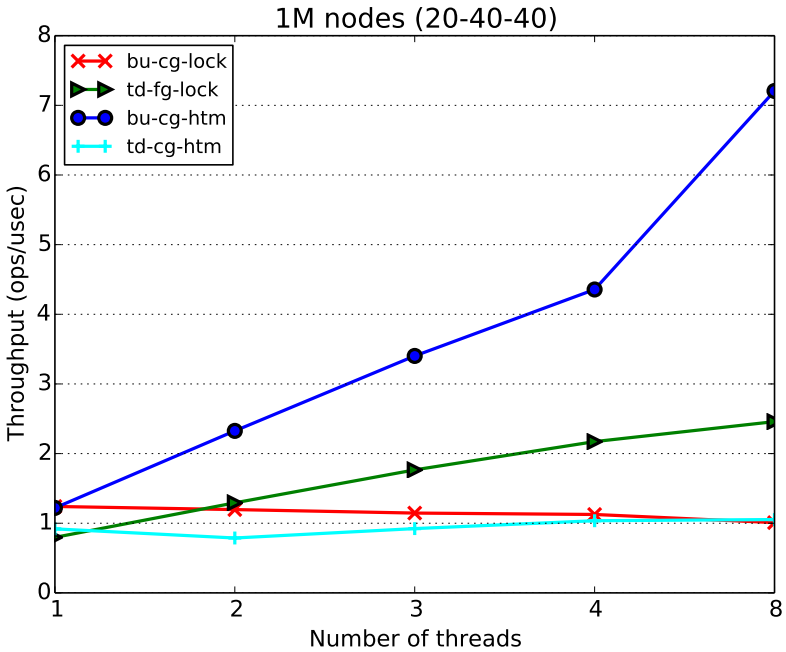


Power8

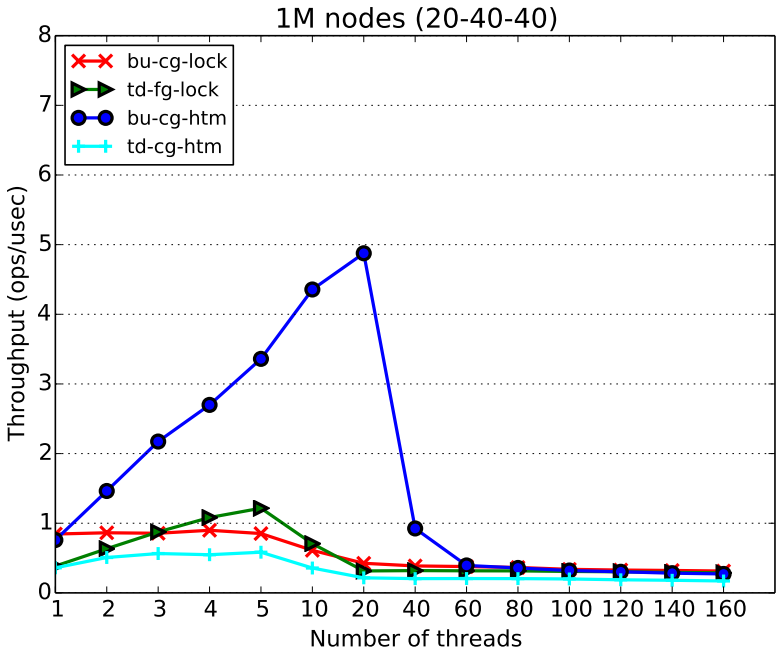


HTM VS Locks: 1M nodes (20-40-40)

Haswell

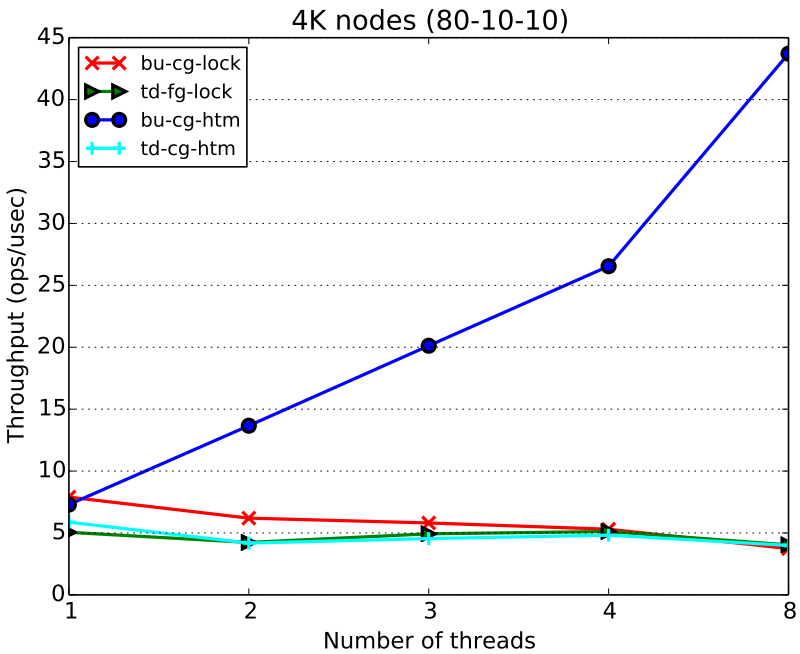


Power8

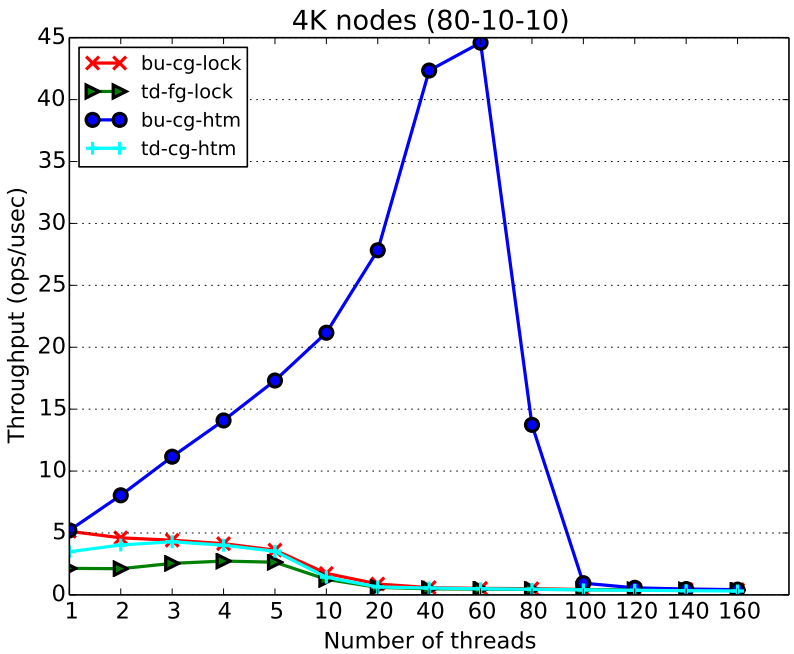


HTM VS Locks: 4K nodes (80-10-10)

Haswell

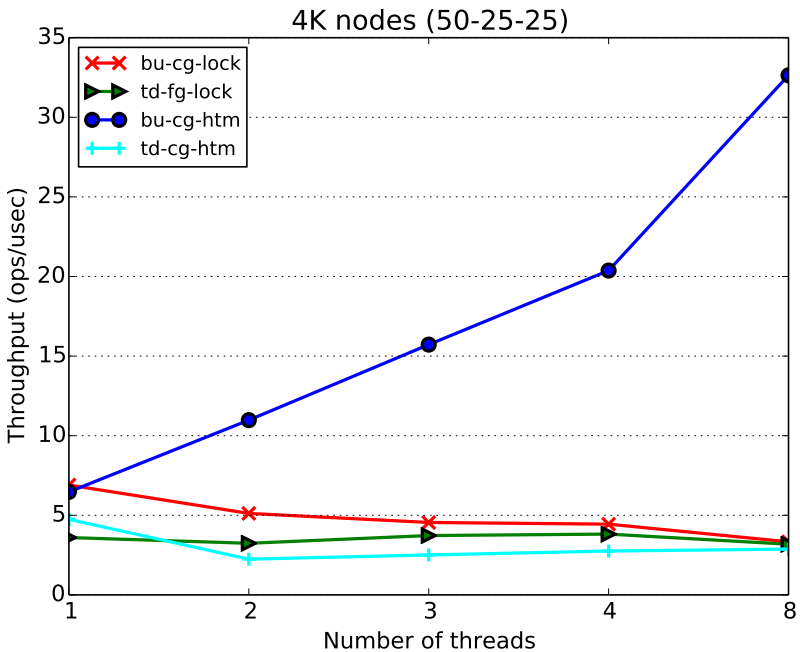


Power8

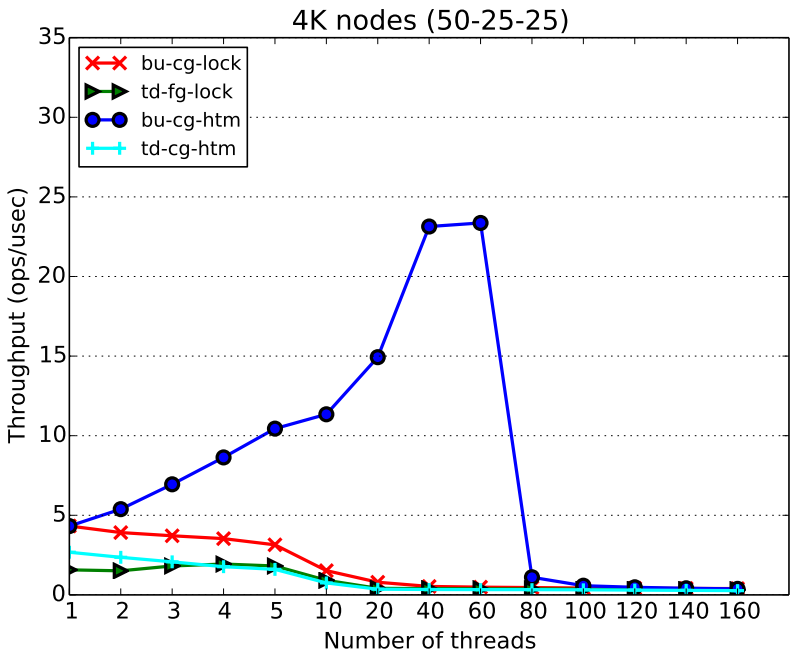


HTM VS Locks: 4K nodes (50-25-25)

Haswell

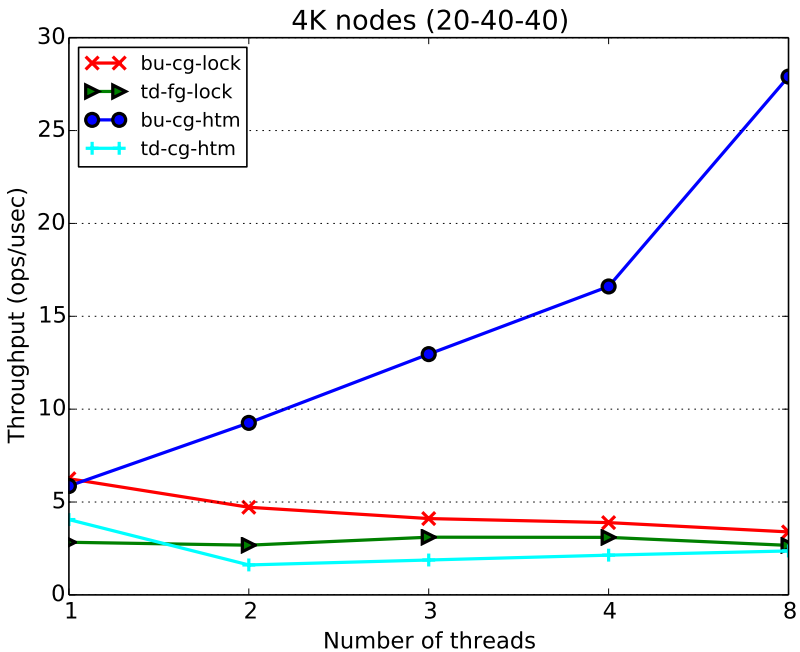


Power8

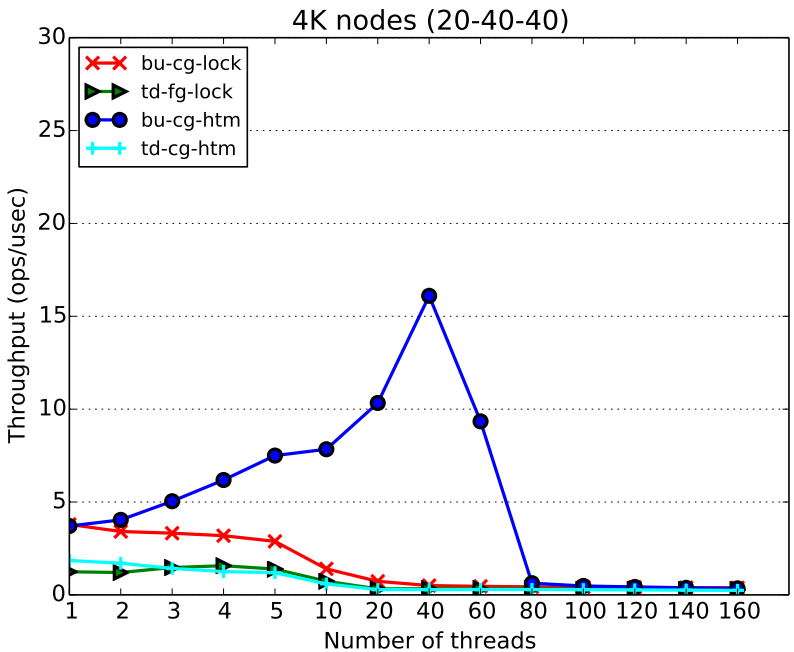


HTM VS Locks: 4K nodes (20-40-40)

Haswell

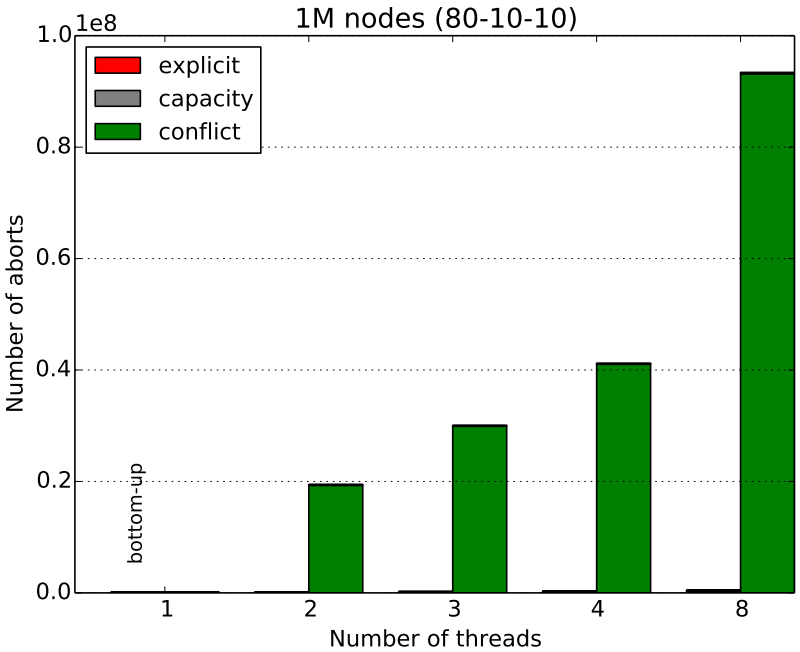


Power8

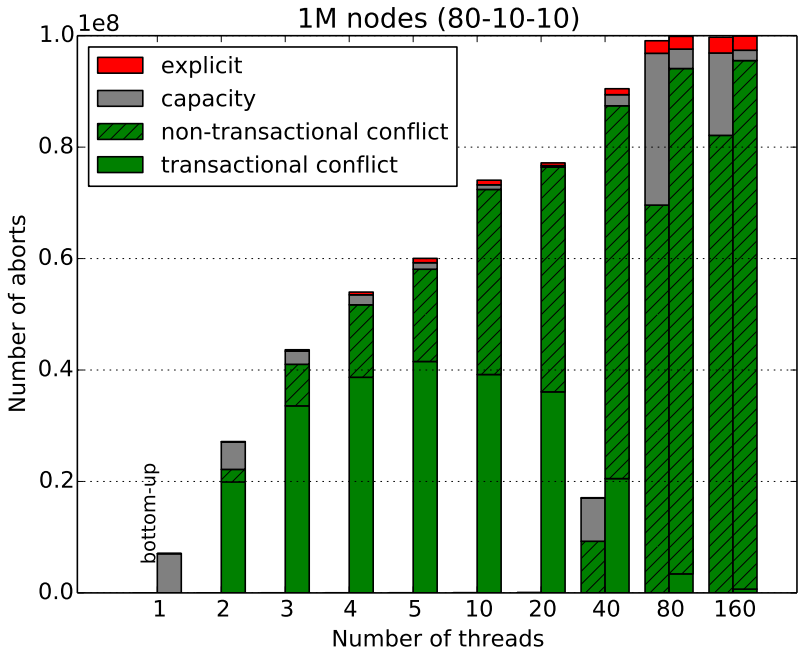


Bottom-up VS Top-down: 1M nodes (80-10-10)

Haswell

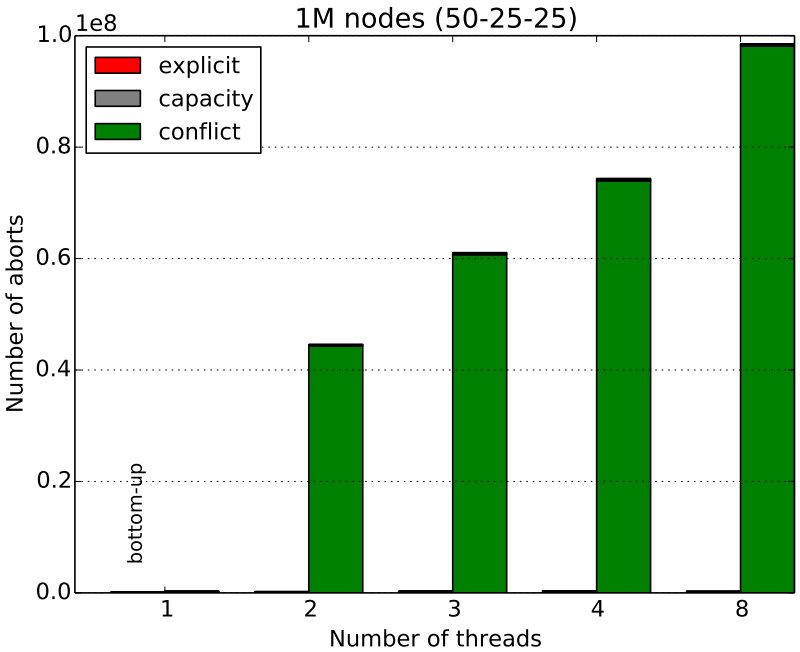


Power8

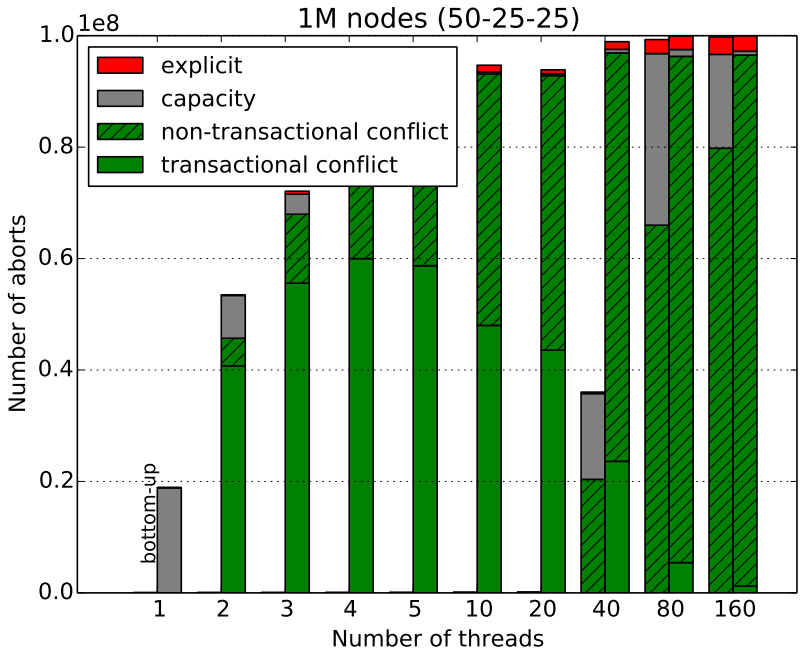


Bottom-up VS Top-down: 1M nodes (50-25-25)

Haswell

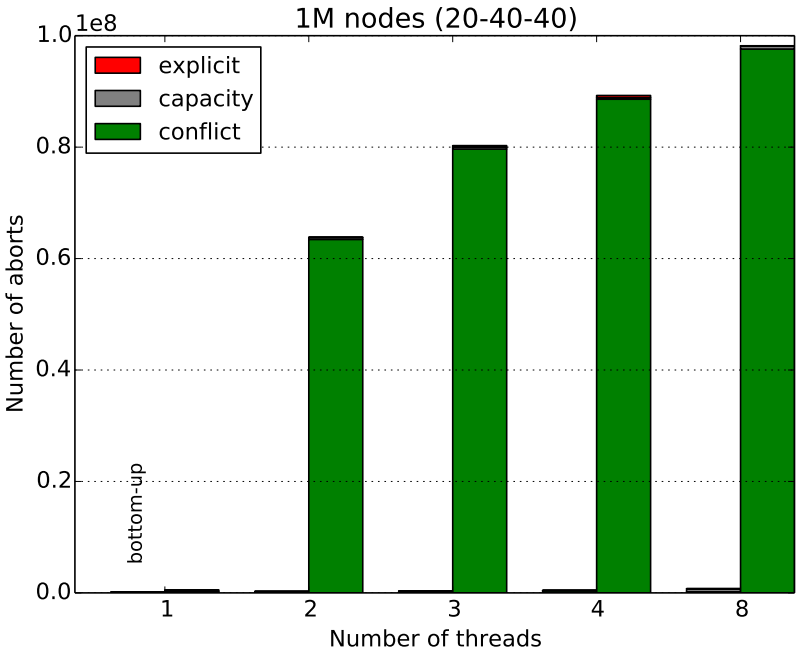


Power8

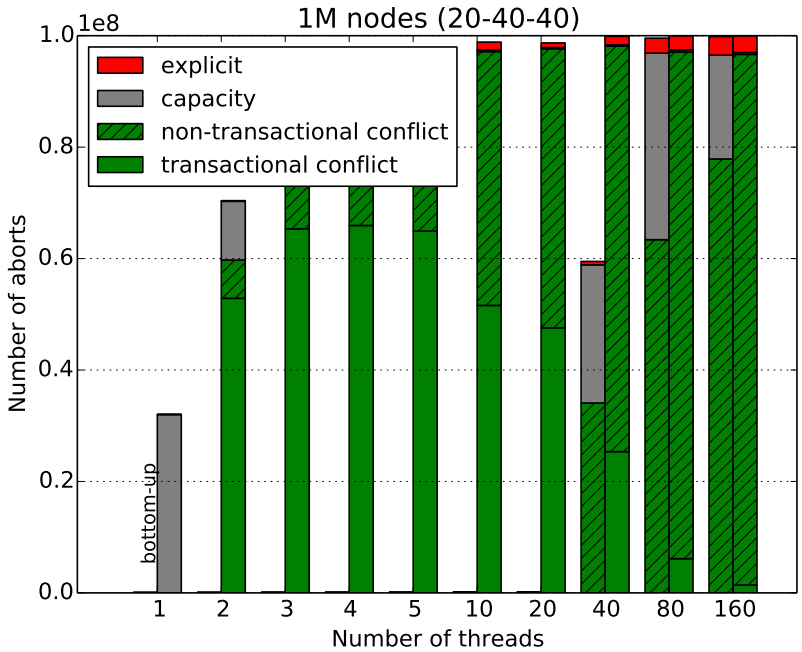


Bottom-up VS Top-down: 1M nodes (20-40-40)

Haswell

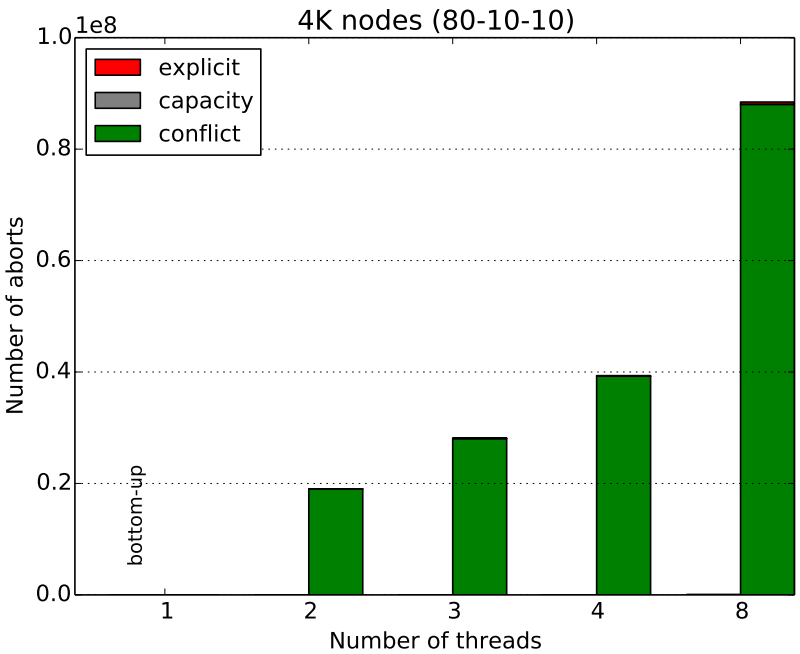


Power8

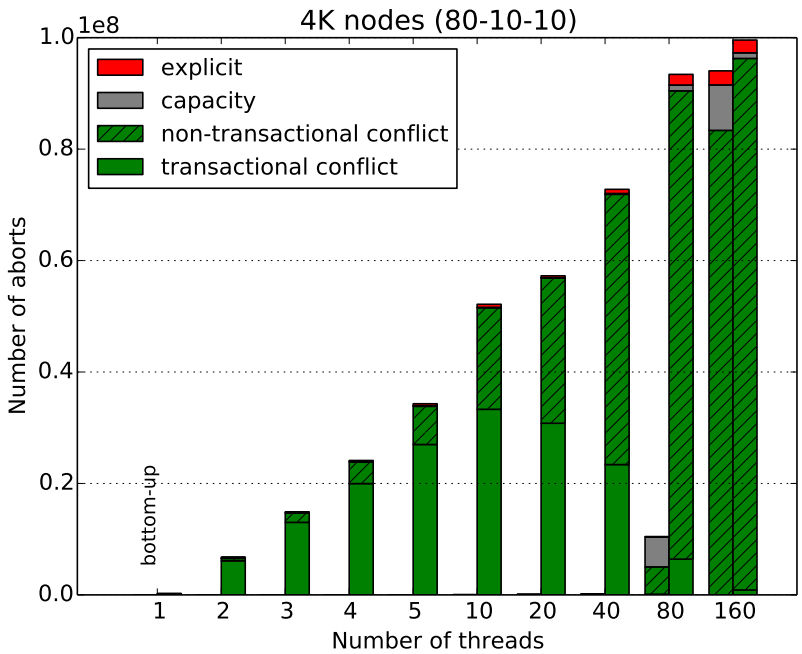


Bottom-up VS Top-down: 4K nodes (80-10-10)

Haswell

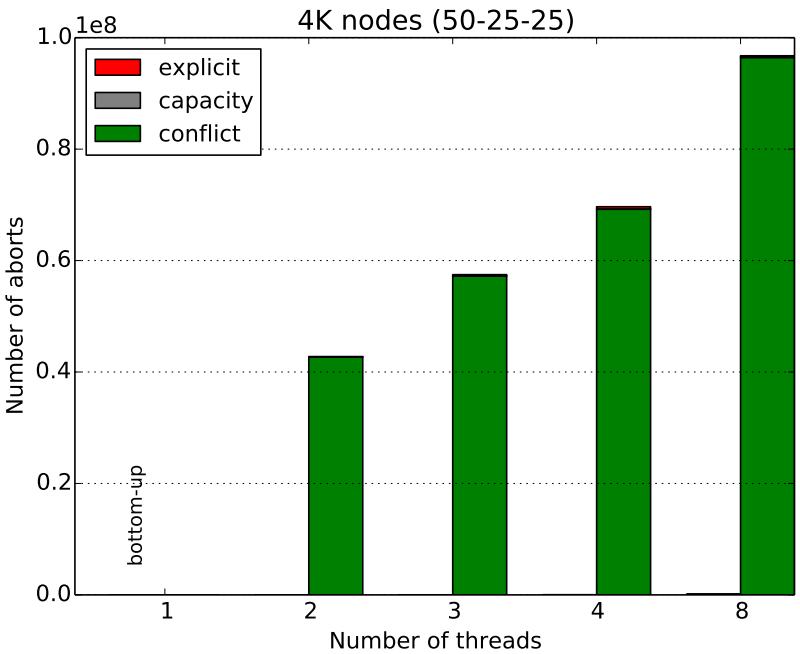


Power8

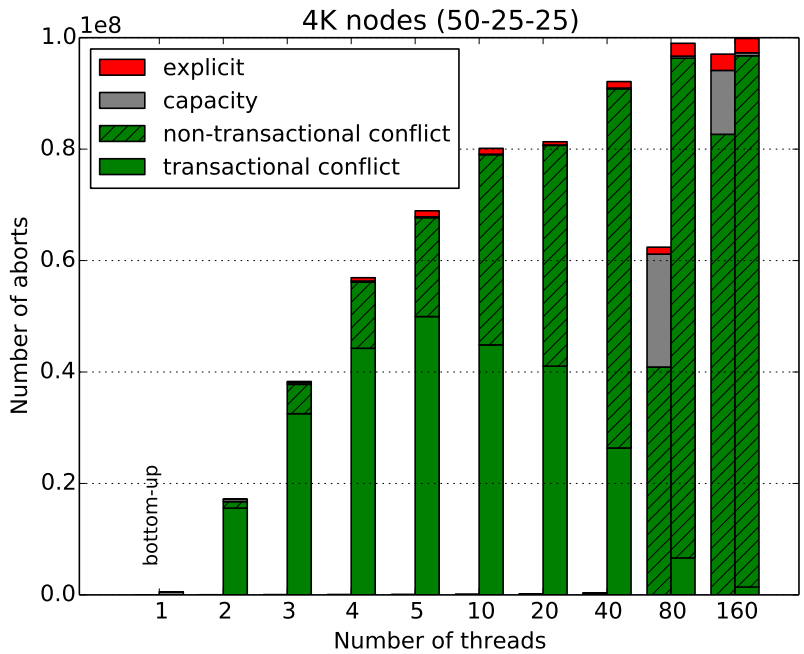


Bottom-up VS Top-down: 4K nodes (50-25-25)

Haswell

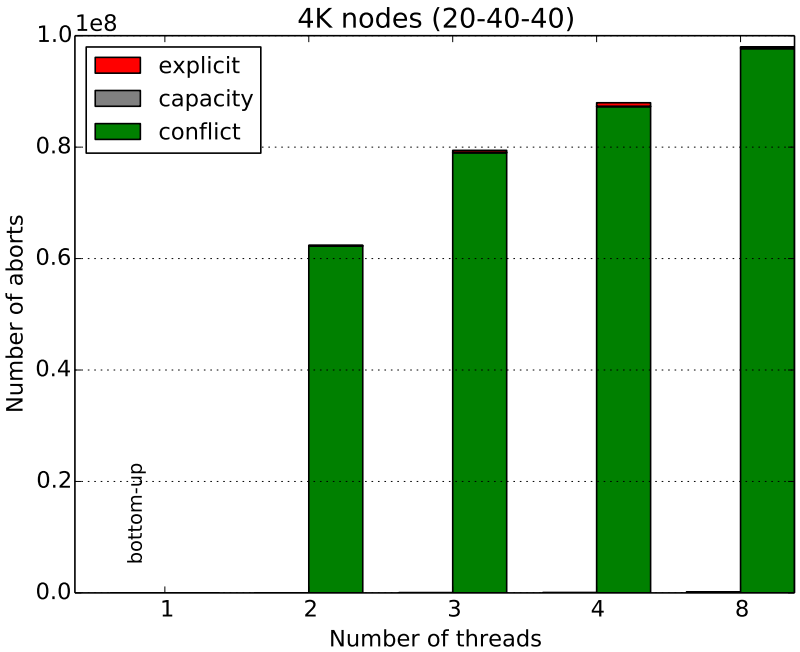


Power8

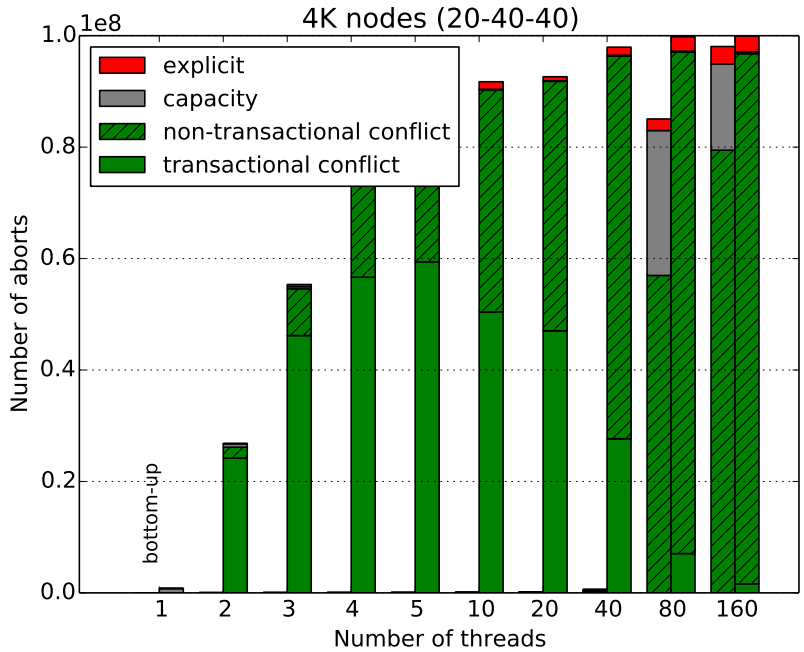


Bottom-up VS Top-down: 4K nodes (20-40-40)

Haswell



Power8



Haswell VS Power8 single threaded performance

- Bottom-up serial, 80-10-10 workload.

Tree Size (Nodes)	**Haswell**	**Power8**
1K	11.82	6.82
1M	1.38	0.97
10M	0.86	0.53

Table : Single thread throughput (ops/usec).

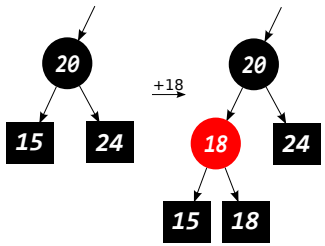
Number of recolors per fixup operation

Recolors	Nodes			
	2M	20M	100M	200M
0	77.12%	75.96%	73.37%	73.08%
1	19.52%	18.66%	18.73%	18.79%
2	2.78%	3.69%	5.30%	5.47%
>2	0.56%	1.69%	2.59%	2.66%

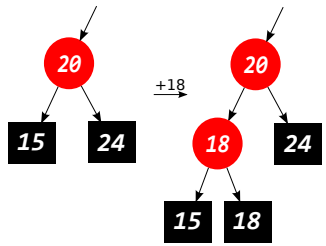
Table : Distribution of fix-ups based on performed recolors.

Insertion

- When the newly inserted node's parent is red, a red-red violation is created:

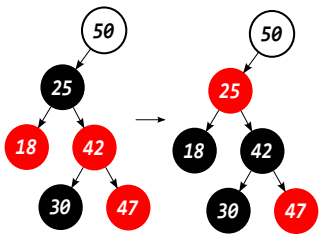


No violation

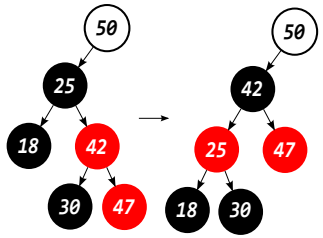


Red-red violation

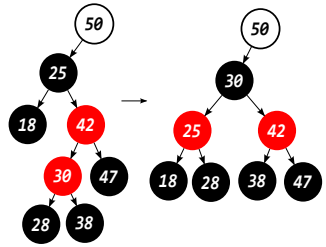
- When a violation is created a fixup is required:



Recolor



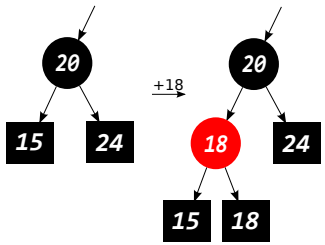
Single rotation



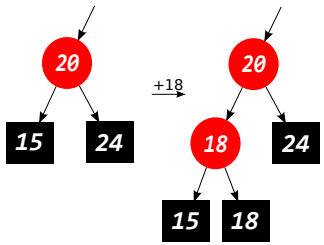
Double rotation

Deletion

- When the removed internal node is black the path becomes short:

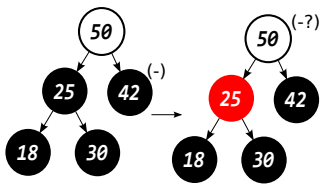


No violation

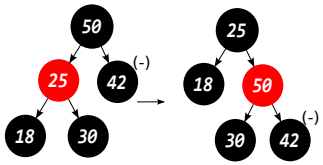


Red-red violation

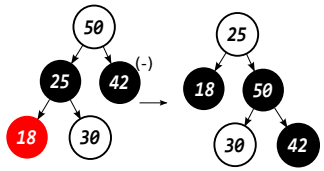
- When a violation is created a fixup is required:



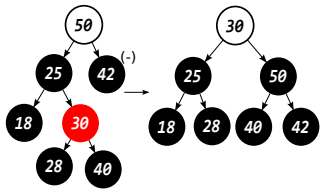
Recolor



Recolor



Recolor



Recolor