Transactional Interference-less Balanced Tree

Ahmed Hassan, Roberto Palmieri, Binoy Ravindran

Systems Software Research Group

Virginia Tech

TRANSACT 2015

Virginia Tech

Concurrent Balanced Trees

- Wide spectrum in literature
  - Lock-Based
    - Traversal: Hand-over-hand-locking, unmonitored, ...
    - Balancing: strict, relaxed
  - Non-Blocking
    - Obstruction-free, lock-free, wait-free...

An Orthogonal Extension

- Transactional Interference-less Balanced Trees.

An Orthogonal Extension

- Transactional Interference-less Balanced Trees.
  - Transactional: Functionality (e.g. Stamp).

An Orthogonal Extension

- Transactional Interference-less Balanced Trees.
  - Transactional: Functionality (e.g. Stamp).
  - Interference-less: Performance.

Transactional Interference-less Balanced Tree

Transactional Interference-less Tree

- Concurrent Trees:
  - Atomic operations: e.g., add; remove; contains.

Transactional Interference-less Tree

- Concurrent Trees:
  - Atomic operations: e.g., add; remove; contains.

- Transactional Trees:

Transactional Interference-less Tree

- Concurrent Trees:
  - Atomic operations: e.g., add; remove; contains.

- Transactional Trees:

```
Shared data: Tree

atomicFoo()
{
        Tree.add(x);
        Tree.add(y);
}
```

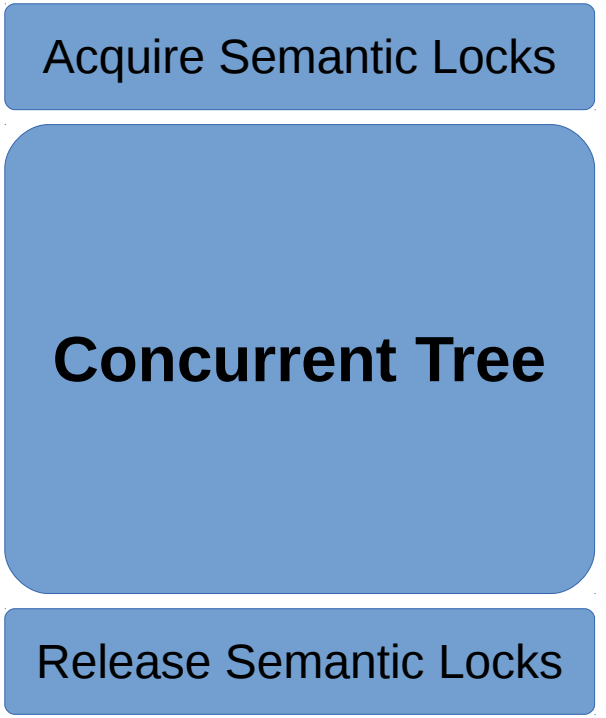
```
Shared data: Tree1, Tree2

atomicFoo()
{
        Tree1.remove(x);
        Tree2.add(x);
}
```

Composability

Solutions?

Solutions?

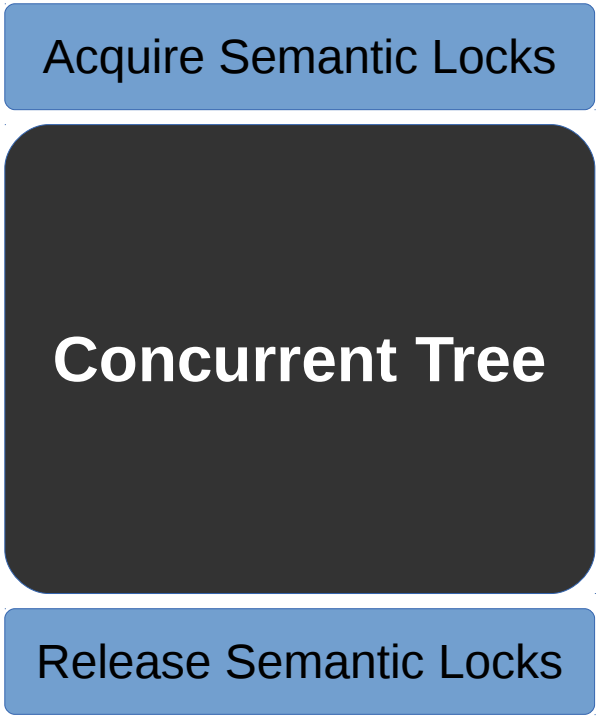
Transactional Memory



Transactional Boosting

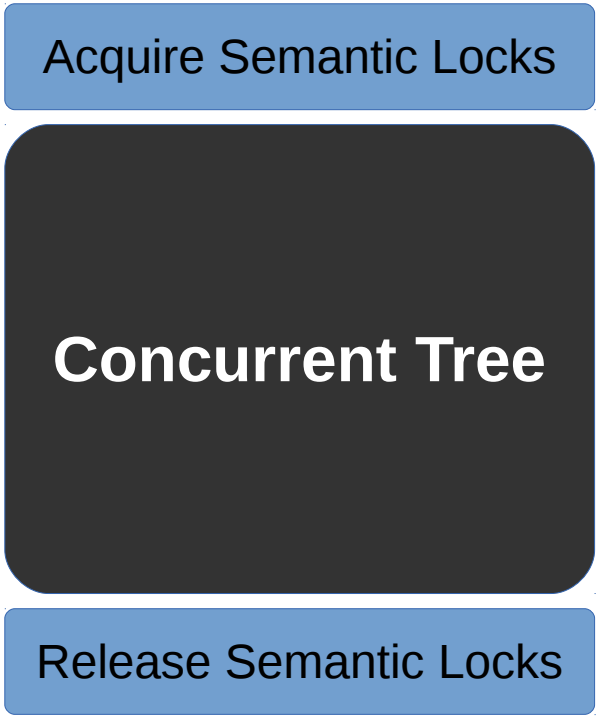
Solutions?

Transactional Memory



Transactional Boosting

Solutions?



Transactional Memory

Transactional Boosting

General, BUT not optimized.

Another Solution:
Optimistic Semantic Synchronization (OSS)

Examples:

- Partitioned Transactions (ParT).
- Consistency Oblivious Programming (COP).
- Optimistic Transactional Boosting (OTB).

Two Steps of OSS

- Step 1: Split operation.

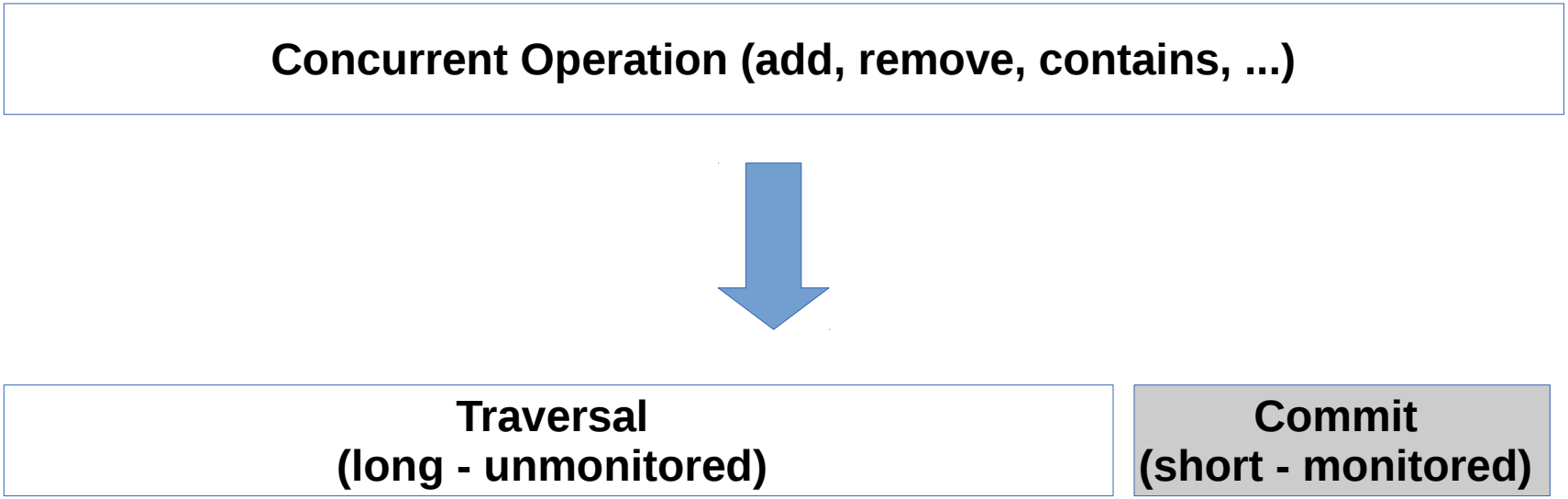
Two Steps of OSS

- Step 1: Split operation.

Concurrent Operation (add, remove, contains, ...)

Two Steps of OSS

- Step 1: Split operation.

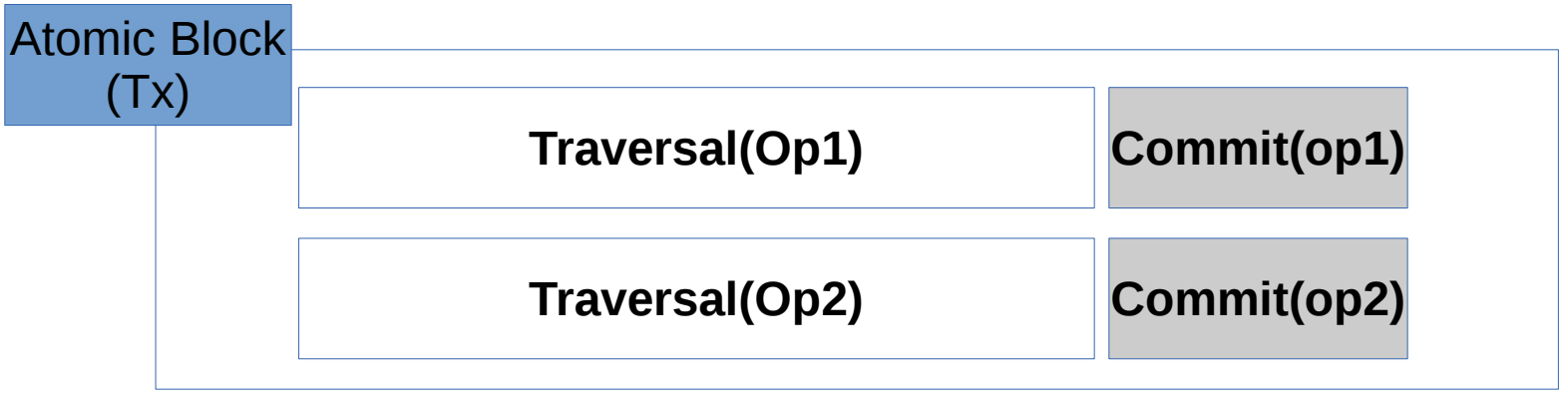


Two Steps of OSS

- Step 2: Compose phases.

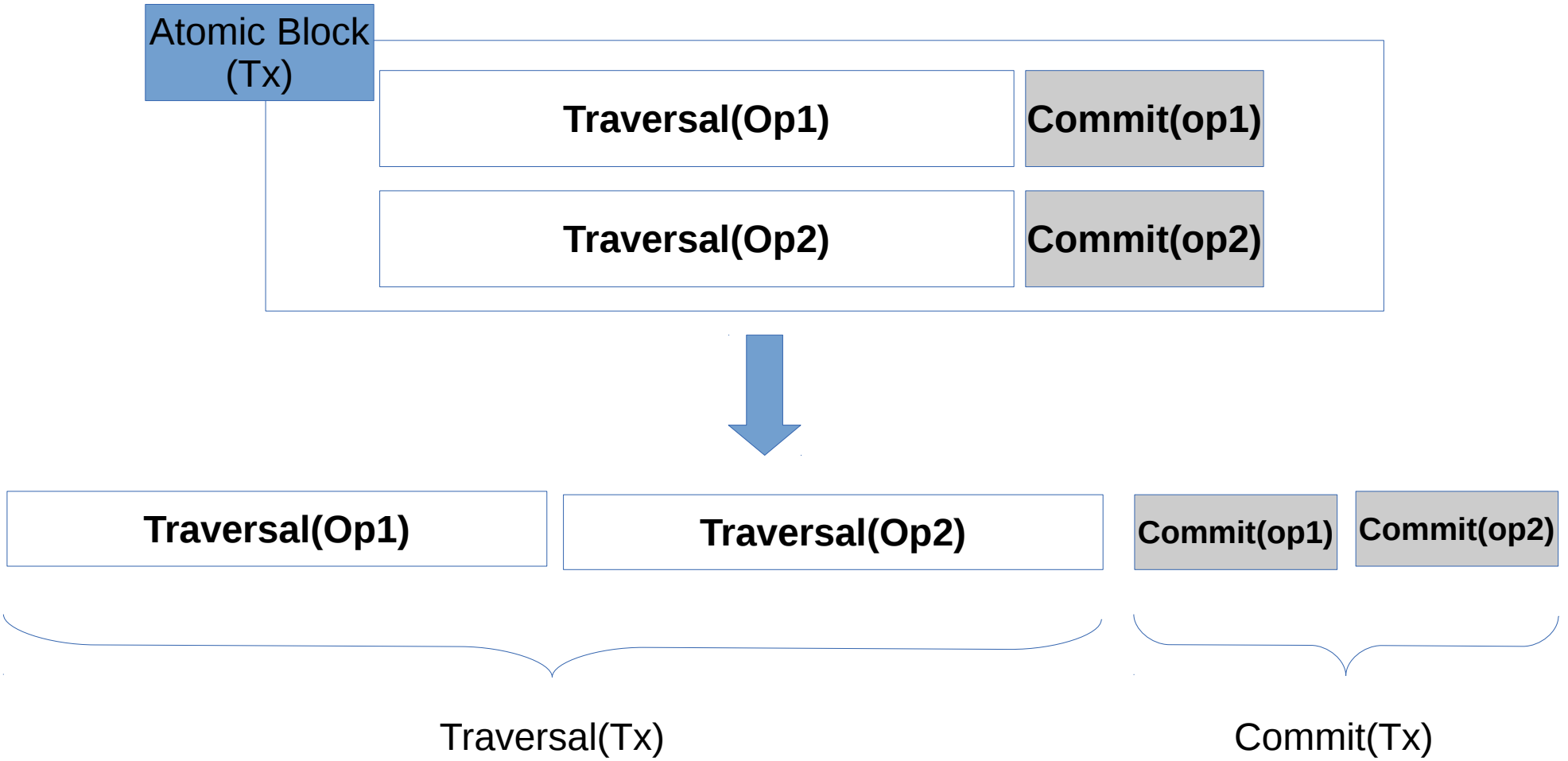
Two Steps of OSS

- Step 2: Compose phases.



Two Steps of OSS

- Step 2: Compose phases.



Low-Level Details

- How to commit (abstract locks, TM, …).
- How to validate.
- How to handle dependent operations in the same transaction.

OSS Vs General Approaches

- White-Boxes Vs Black-Boxes
- Optimization Vs Generality.
- Both extend “existing” implementations.

The Next Question

- Which concurrent balanced tree design fits OSS?

The Next Question

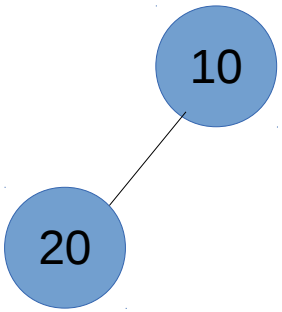
- Which concurrent balanced tree design fits OSS?

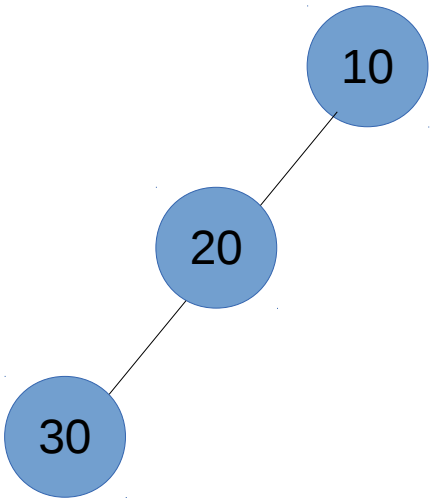
Contention-Friendly Tree

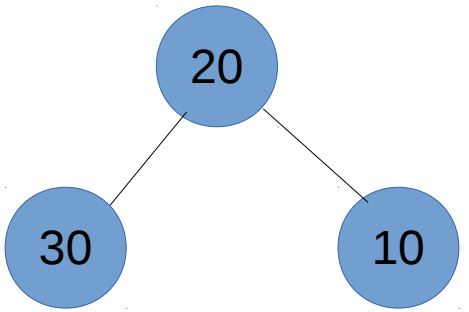
Crain, Gramoli, & Raynal'13

CF-Tree

- Example: Insert 30.

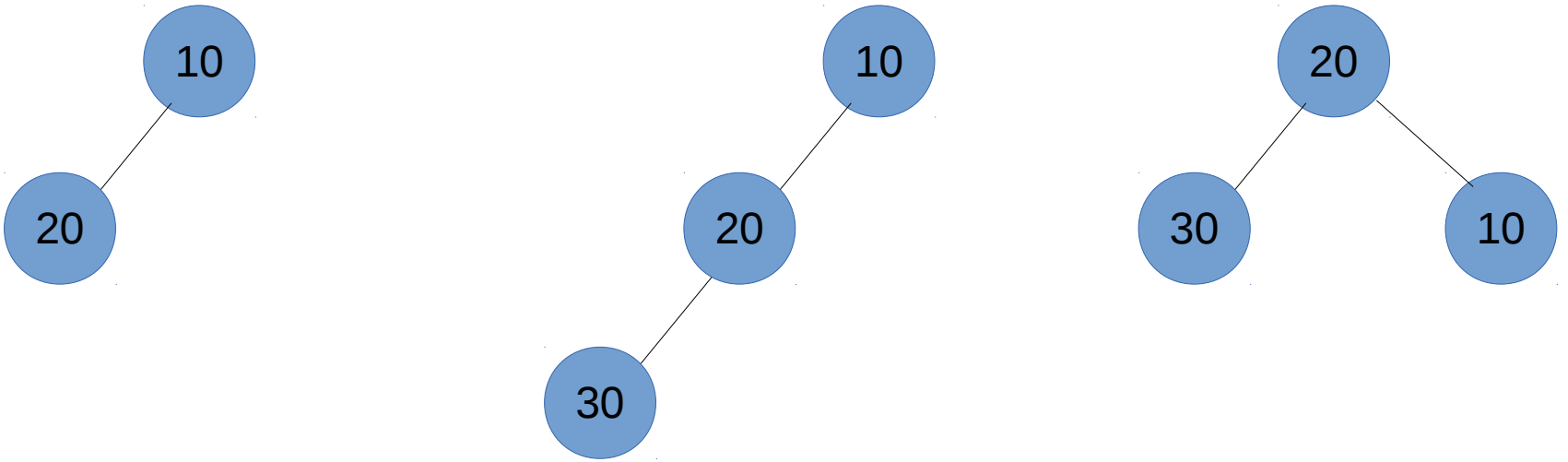


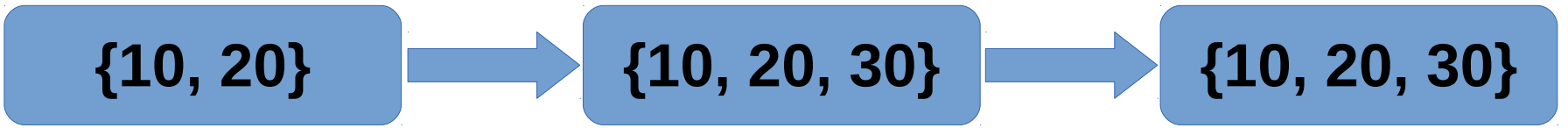




CF-Tree

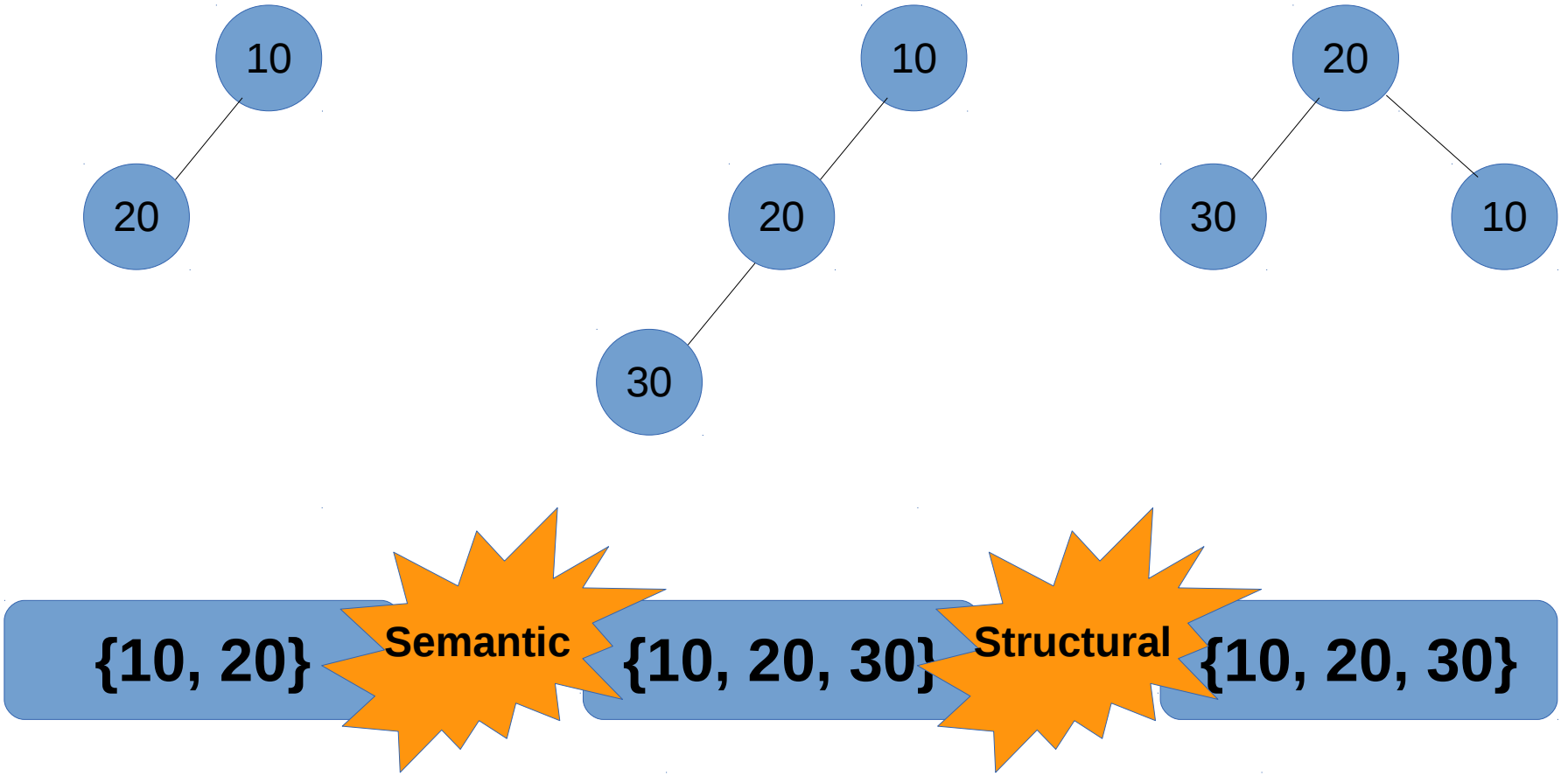
- Example: Insert 30.





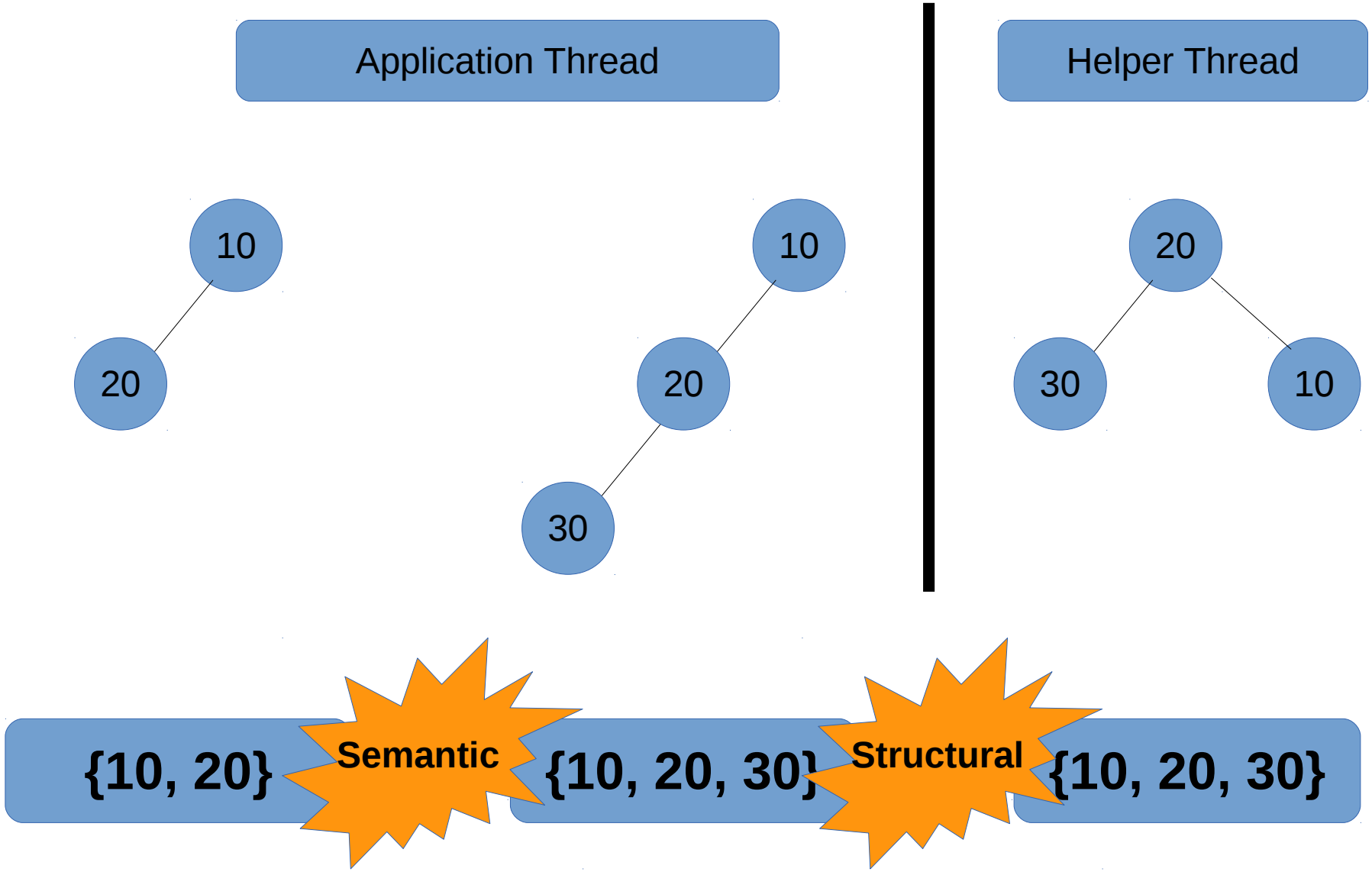
CF-Tree

- Example: Insert 30.



CF-Tree

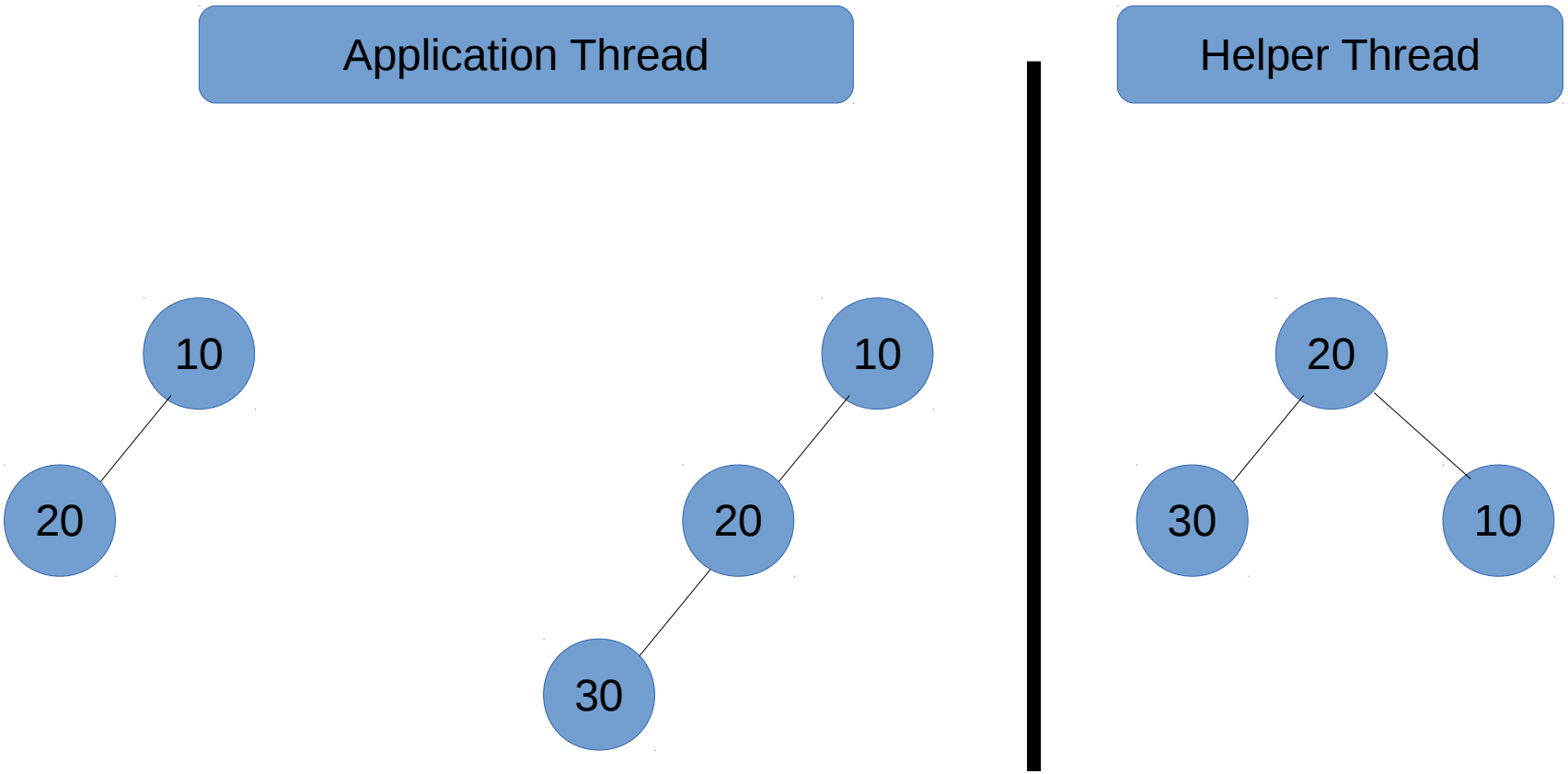
- Example: Insert 30.



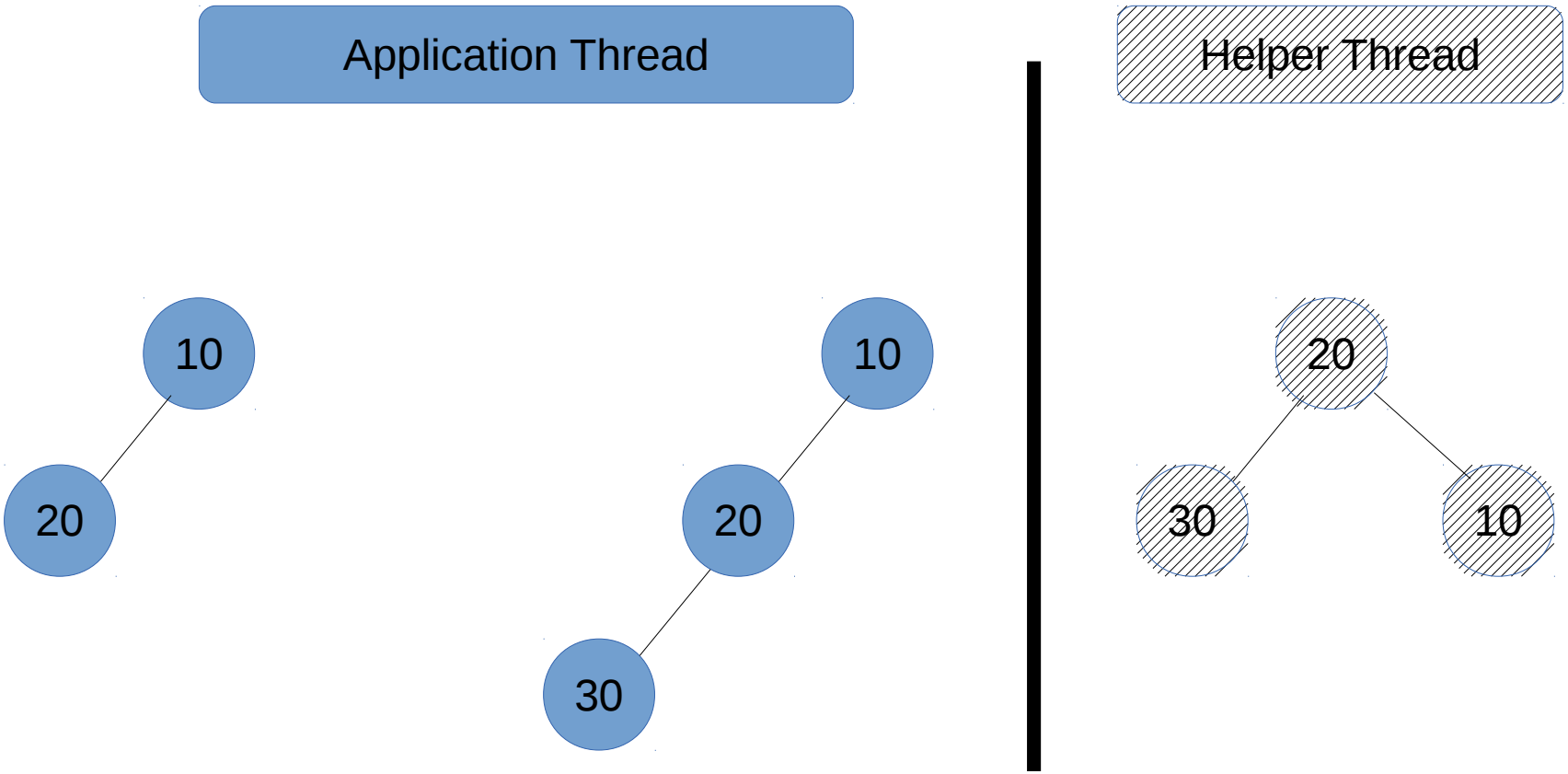
Our Proposal

Transactionalizing CF-Tree using OSS
(TxCF-Tree)

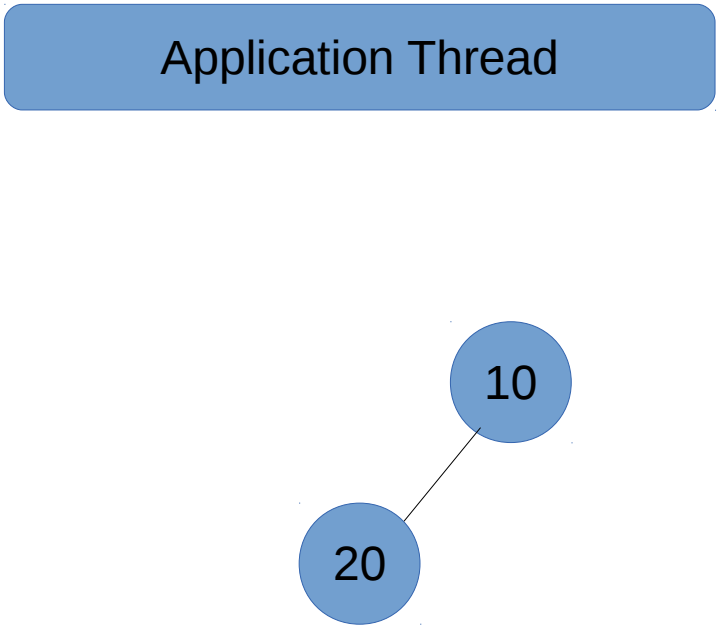
TxCF-Tree



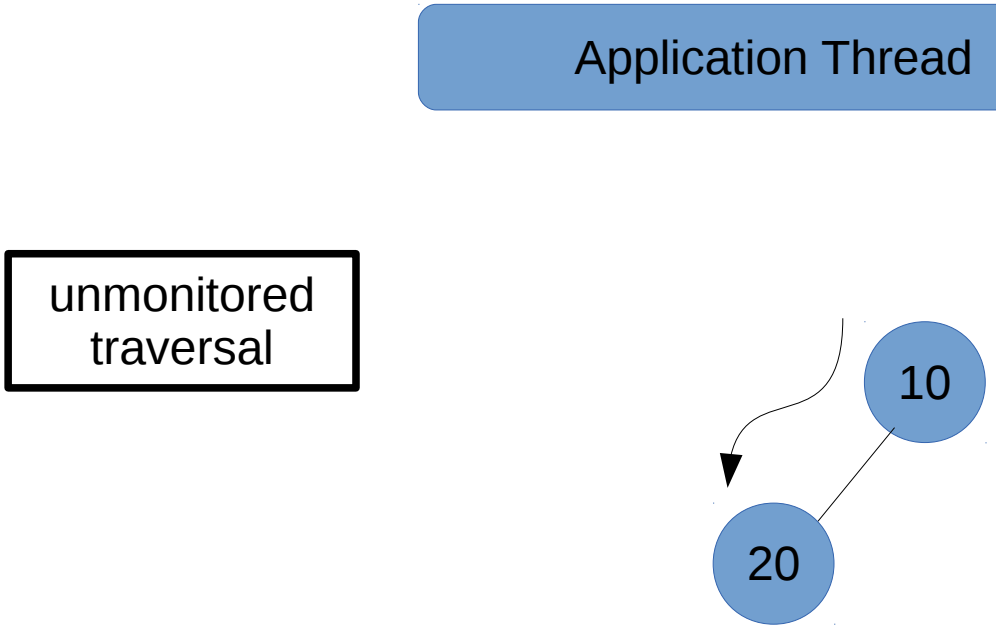
TxCF-Tree



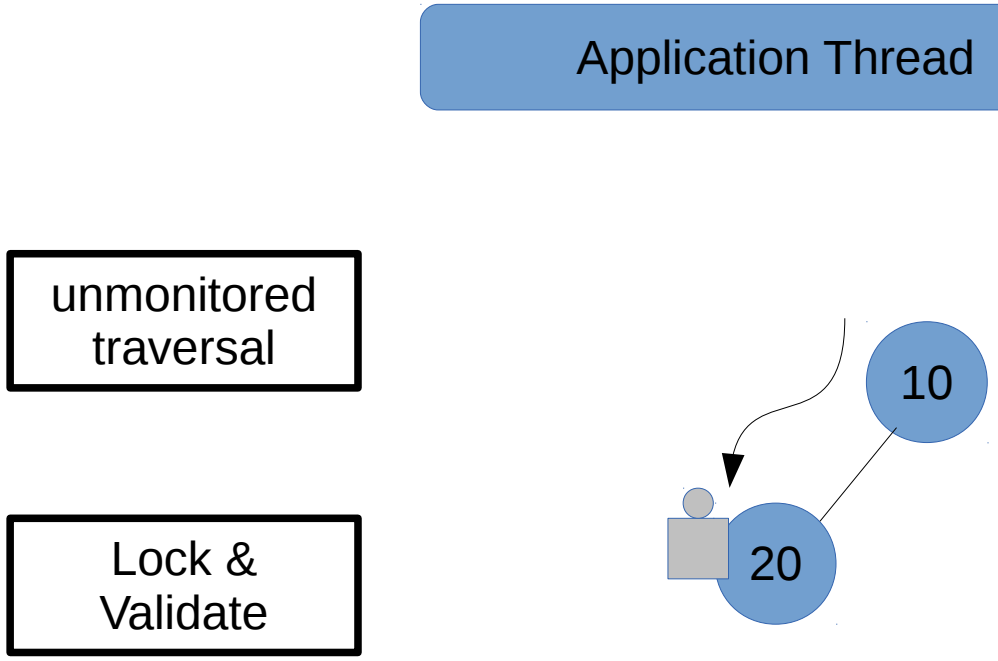
TxCF-Tree



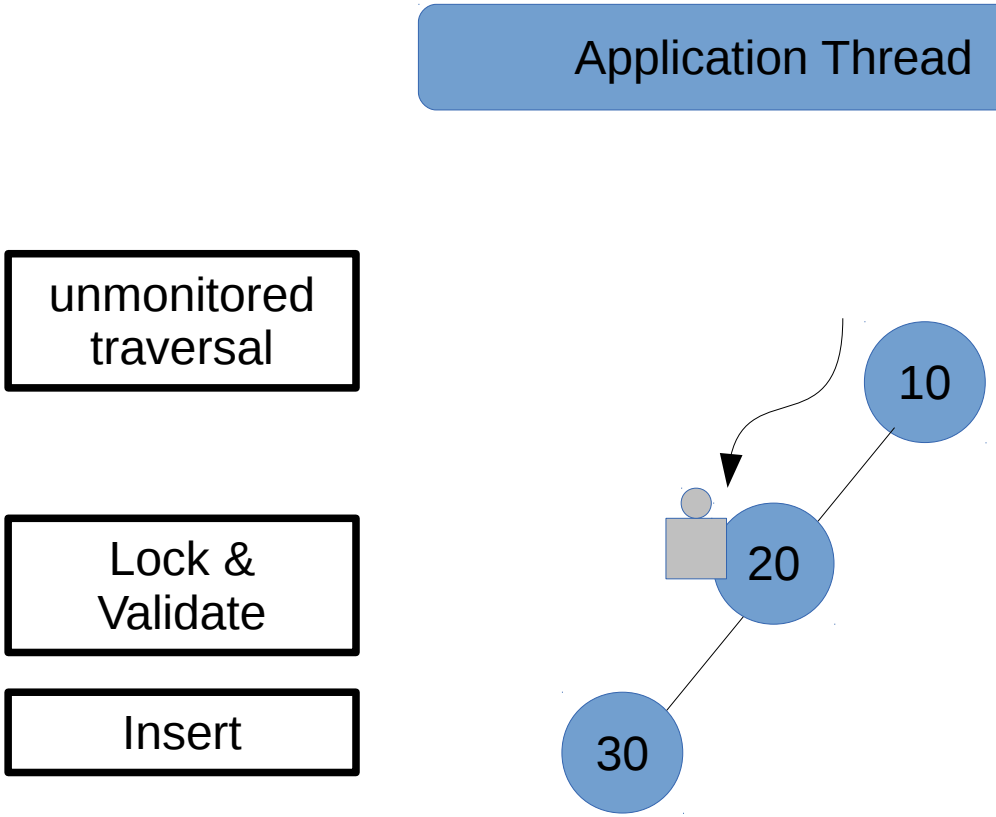
TxCF-Tree



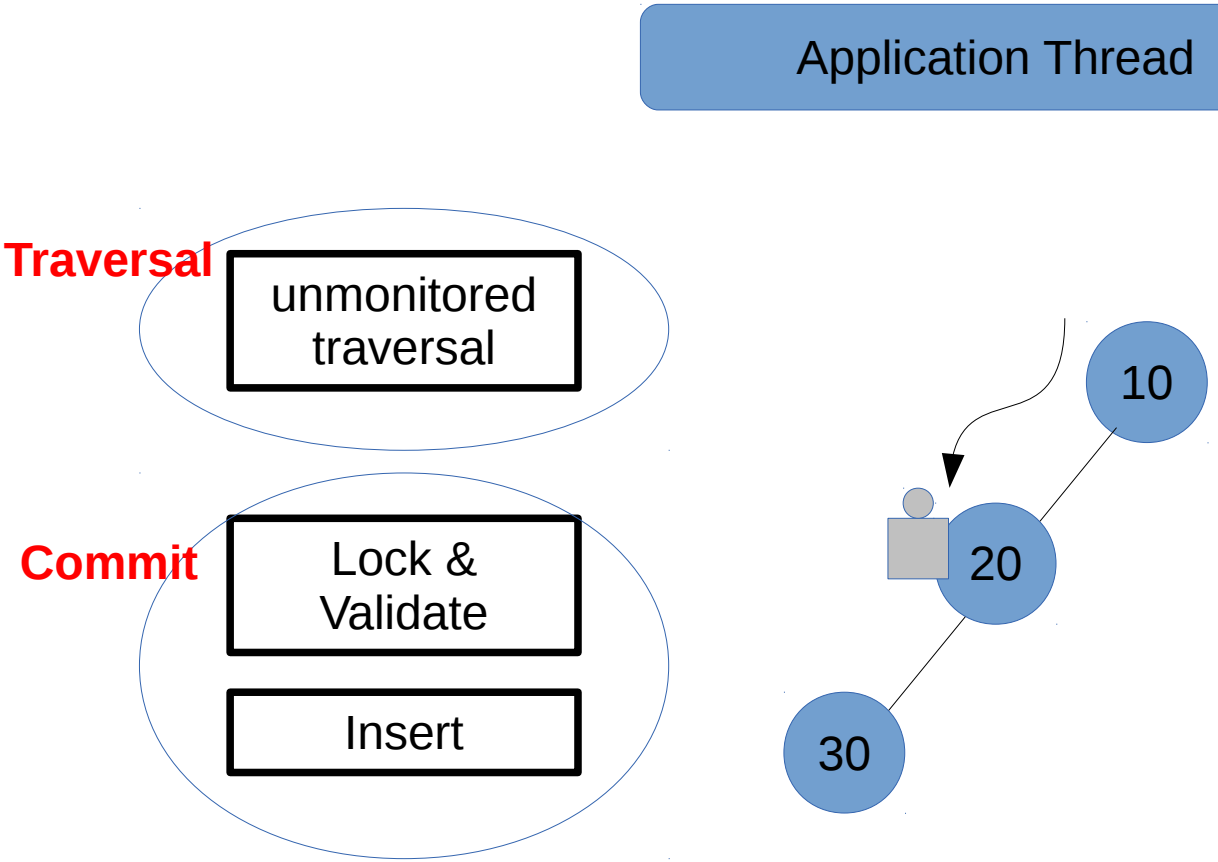
TxCF-Tree


Application Thread
unmonitored traversal
Lock & Validate
10
20

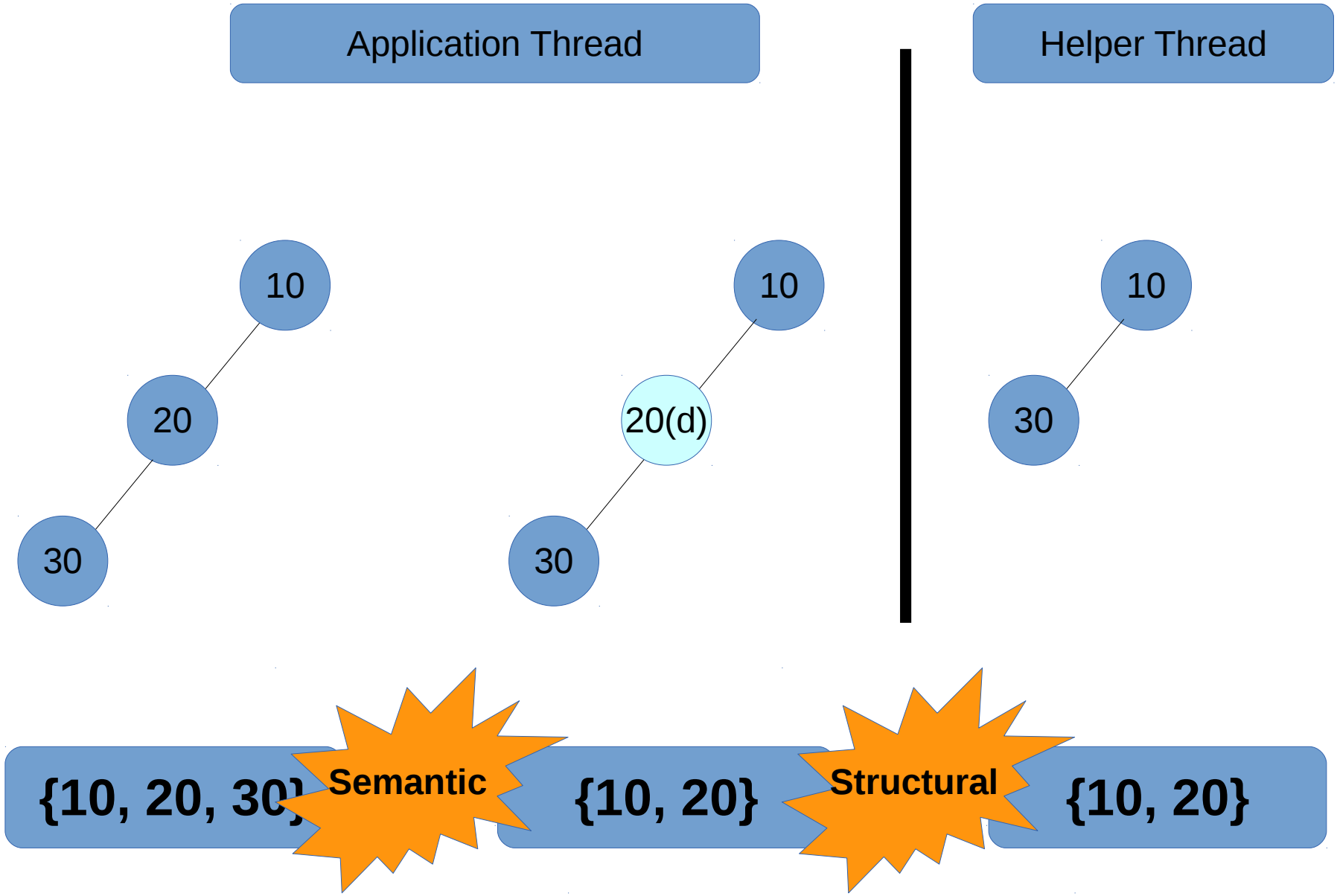
TxCF-Tree


Application Thread
unmonitored traversal
Lock & Validate
Insert
10
20
30

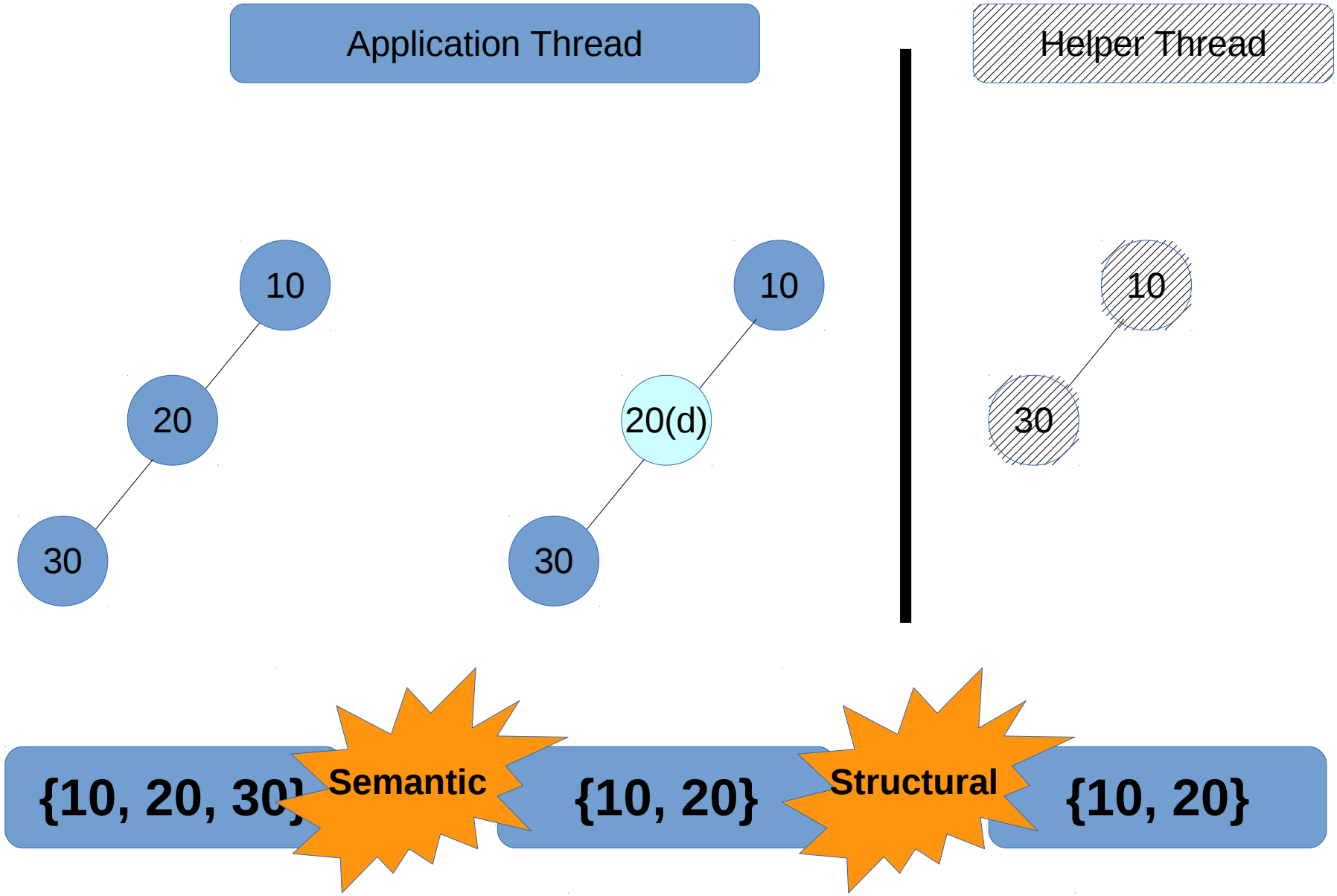
TxCF-Tree


Application Thread
Traversal
unmonitored traversal
Commit
Lock & Validate
Insert
10
20
30

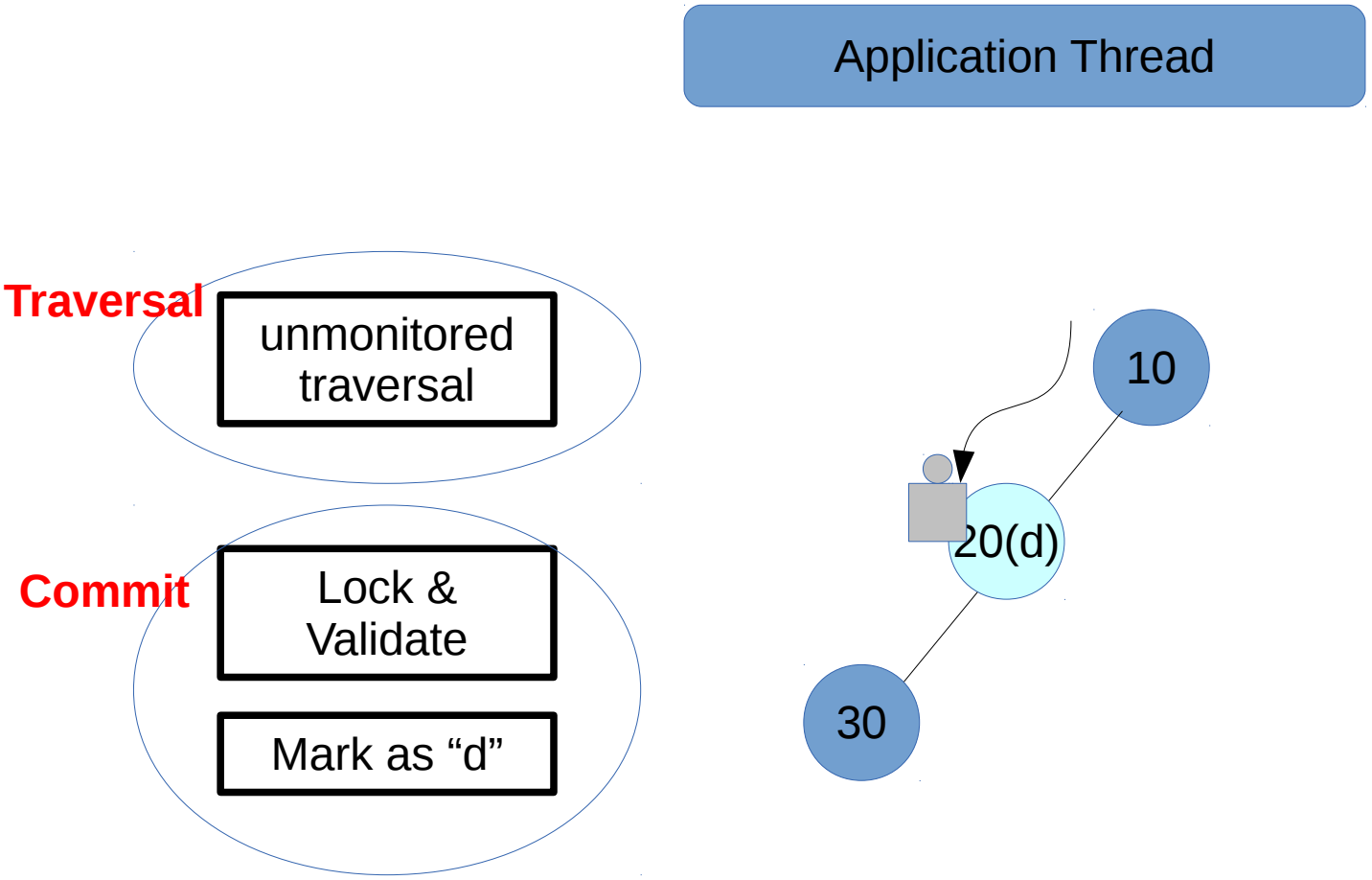
Remove is similar...



Remove is similar...



Remove is similar...



Transactional Interference-less Tree

Transactional Interference-less Tree

- How
  - Step 1: CF-Tree!!
  - Step 2: Always give the highest priority to semantic operations over structural operations.

Transactional Interference-less Tree

- How
  - Step 1: CF-Tree!!
  - Step 2: Always give the highest priority to semantic operations over structural operations.

- Why
  - In concurrent trees: May be less important!!
  - In transactional trees:
    - Aborting transactions rolls back all its operations (including the non-conflicting ones).
    - Long transactions are more prone to interfere with the helper thread.

Three building blocks

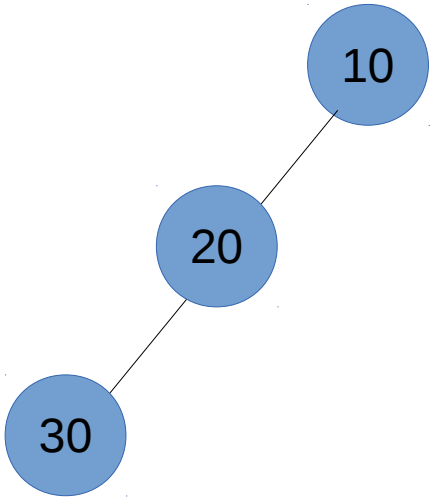
- Structural Locks.
- Structural Invalidation.
- Adaptive Back-off Delay.

Structural Locks

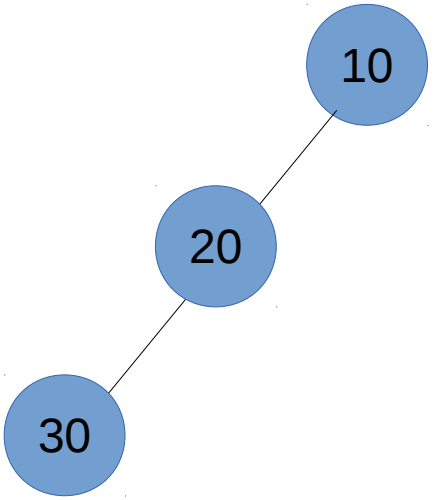
Structural Locks

- Transaction T1 wants to delete 30.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



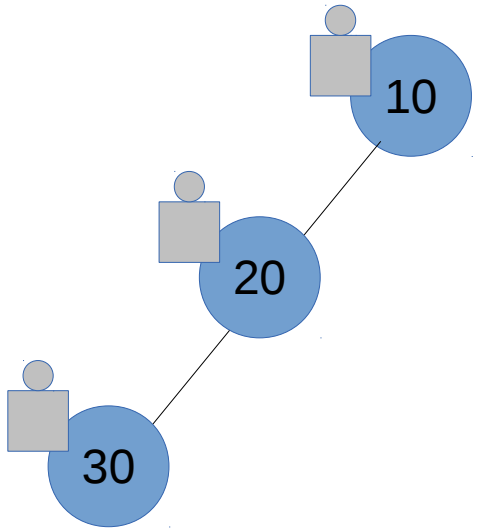
A concurrent delete(30)



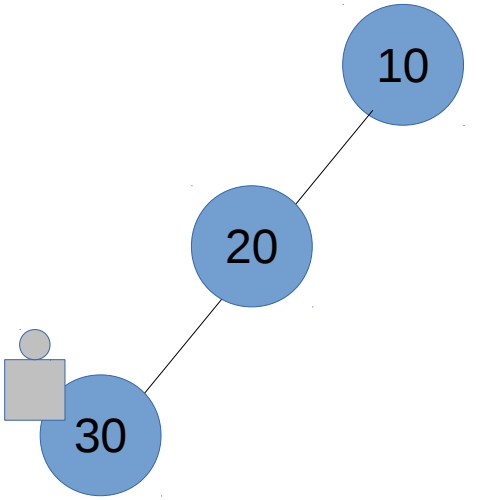
Structural Locks

- Transaction T1 wants to delete 30.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



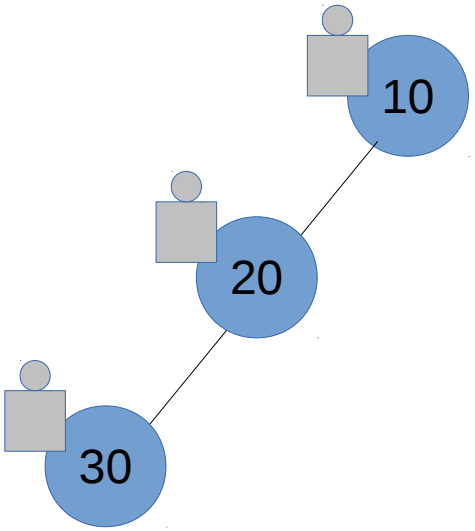
A concurrent delete(30)



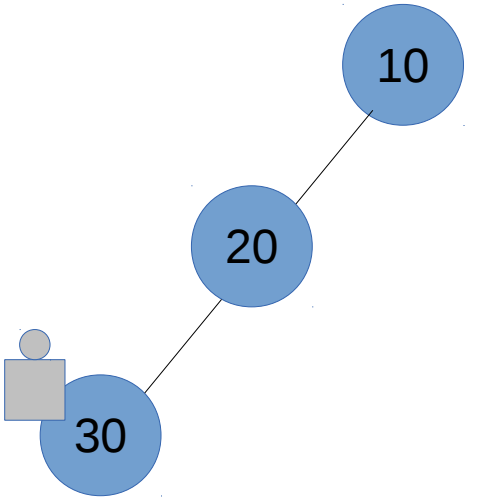
Structural Locks

- Transaction T1 wants to delete 30.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent delete(30)



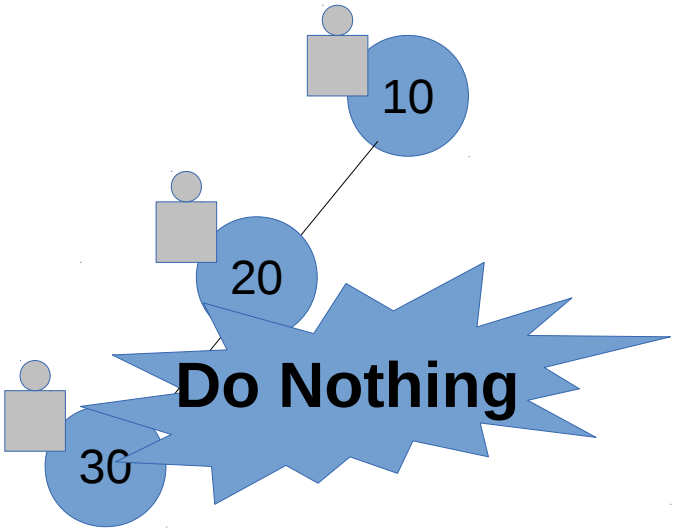
T1 observes that “30” is locked

What is the best to do in both cases?

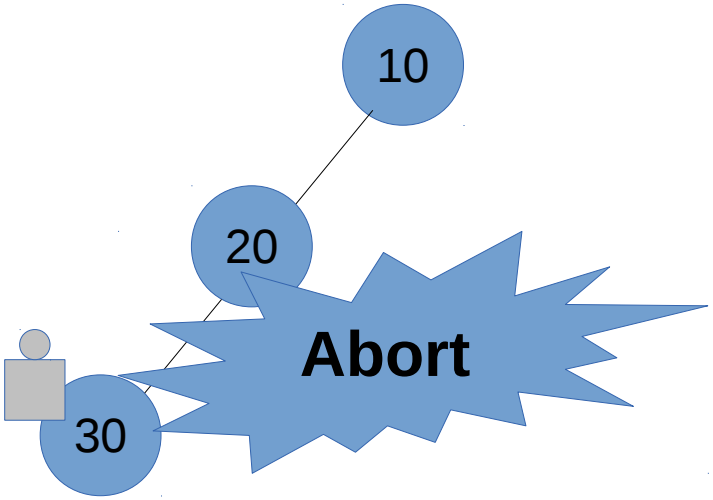
Structural Locks

- Transaction T1 wants to delete 30.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent delete(30)



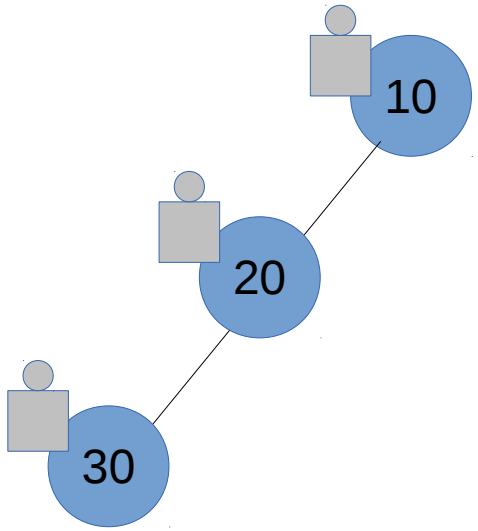
T1 observes that “30” is locked

What is the best to do in both cases?

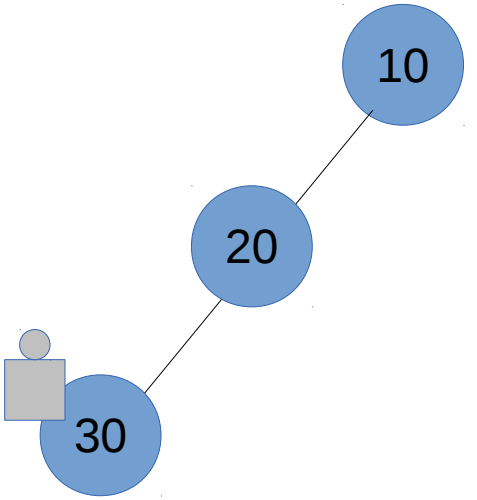
Structural Locks

- Transaction T1 wants to delete 30.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent delete(30)

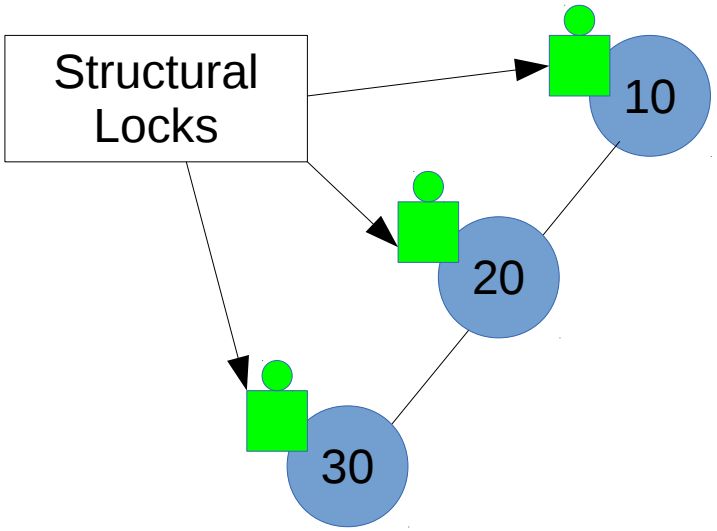


Solution?

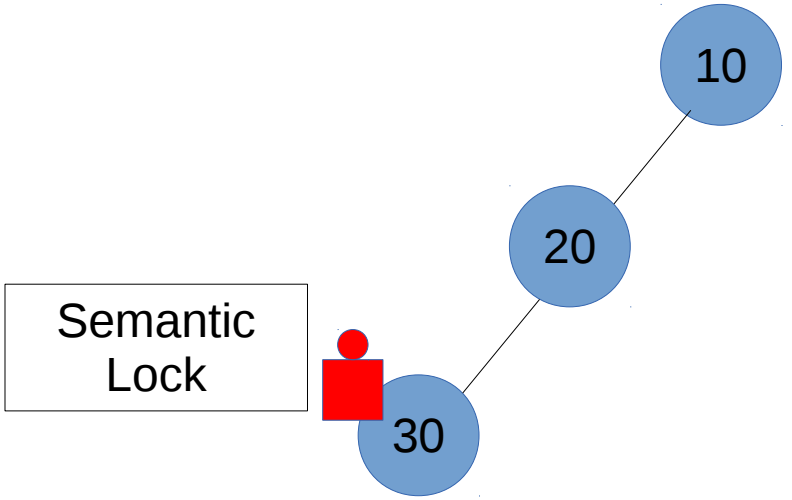
Structural Locks

- Transaction T1 wants to delete 30.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent delete(30)



Solution?

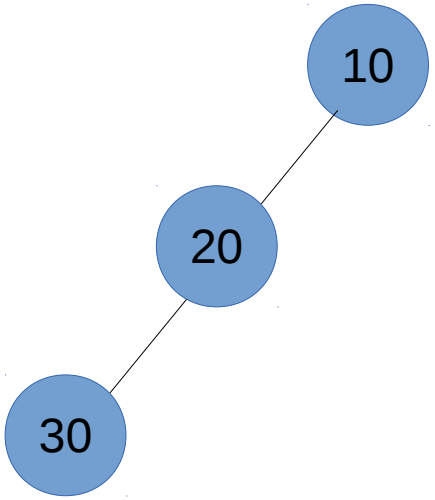
Two types of locks

Structural Invalidation

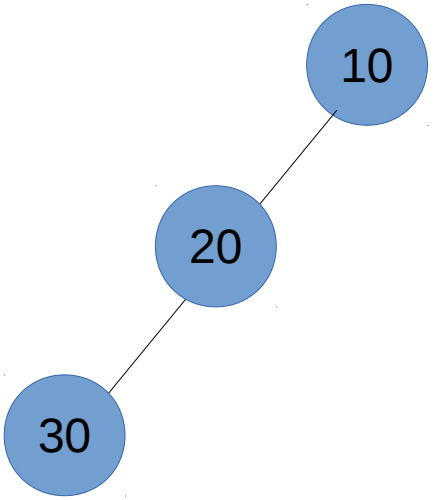
Structural Invalidation

- Transaction T1 wants to insert 15.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



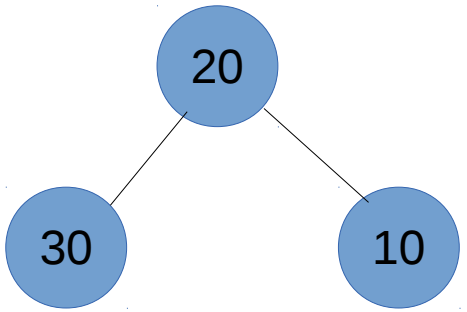
A concurrent insert(15)



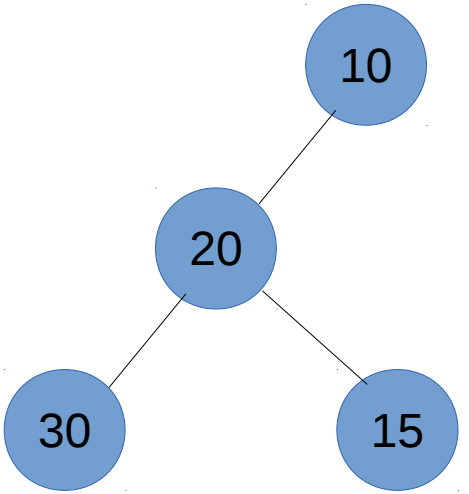
Structural Invalidation

- Transaction T1 wants to insert 15.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



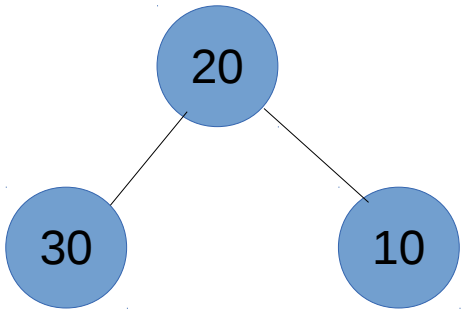
A concurrent insert(15)



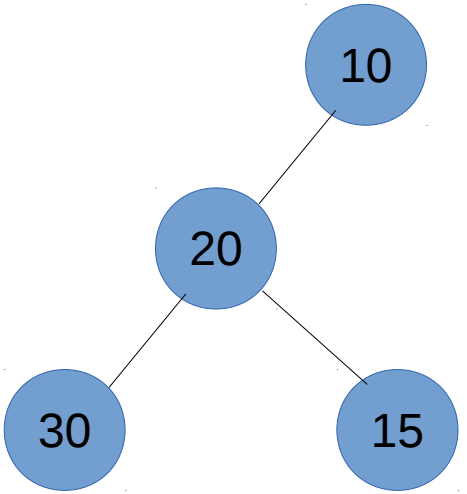
Structural Invalidation

- Transaction T1 wants to insert 15.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent insert(15)



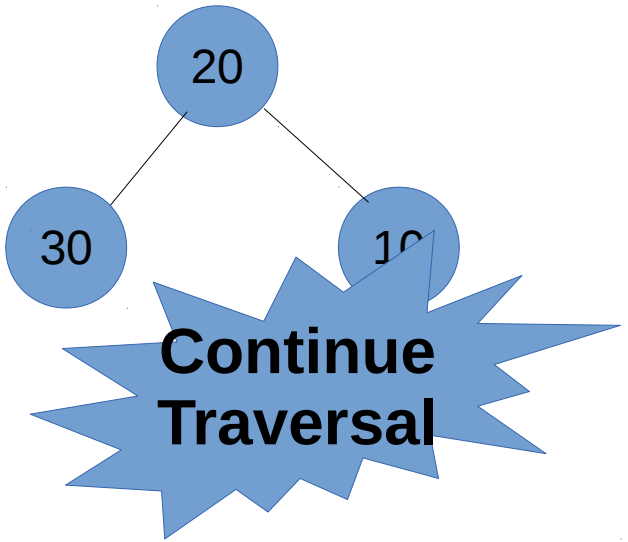
T1 observes that the right child of “20” is not NULL

What is the best to do in both cases?

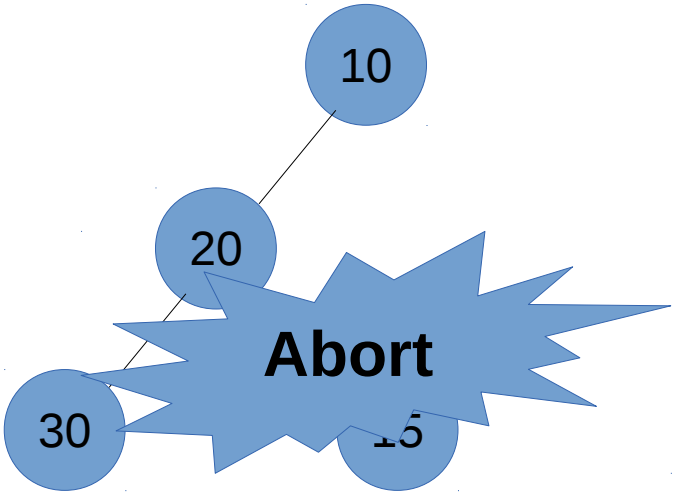
Structural Invalidation

- Transaction T1 wants to insert 15.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent insert(15)



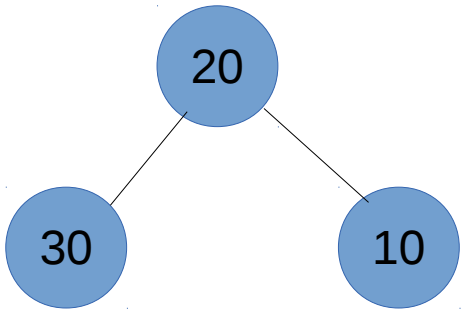
T1 observes that the right child of “20” is not NULL

What is the best to do in both cases?

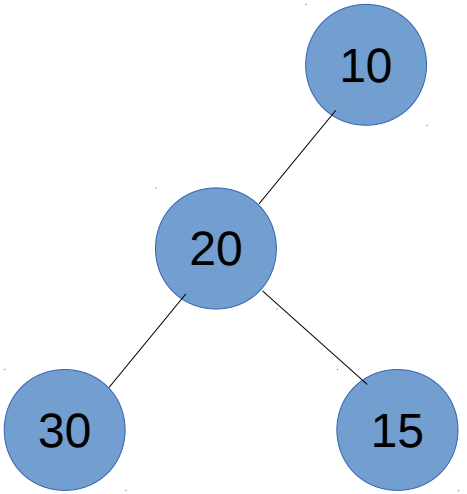
Structural Invalidation

- Transaction T1 wants to insert 15.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent insert(15)

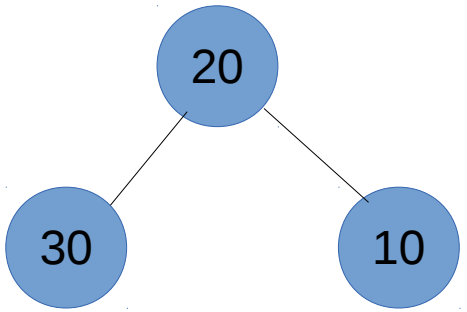


Solution?

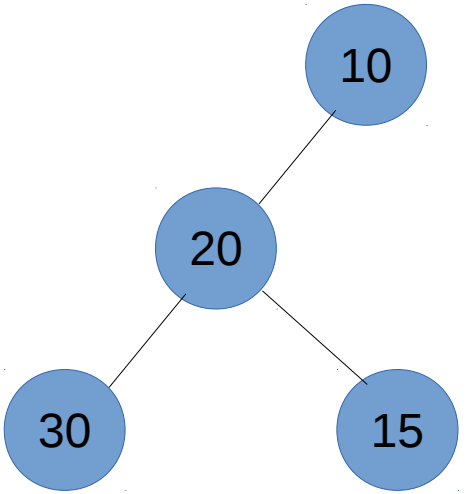
Structural Invalidation

- Transaction T1 wants to insert 15.
- after traversal and before commit, assume 2 scenarios

A concurrent rotation



A concurrent insert(15)



Solution?

Continue Traversal anyway

Adaptive Back-off Delay

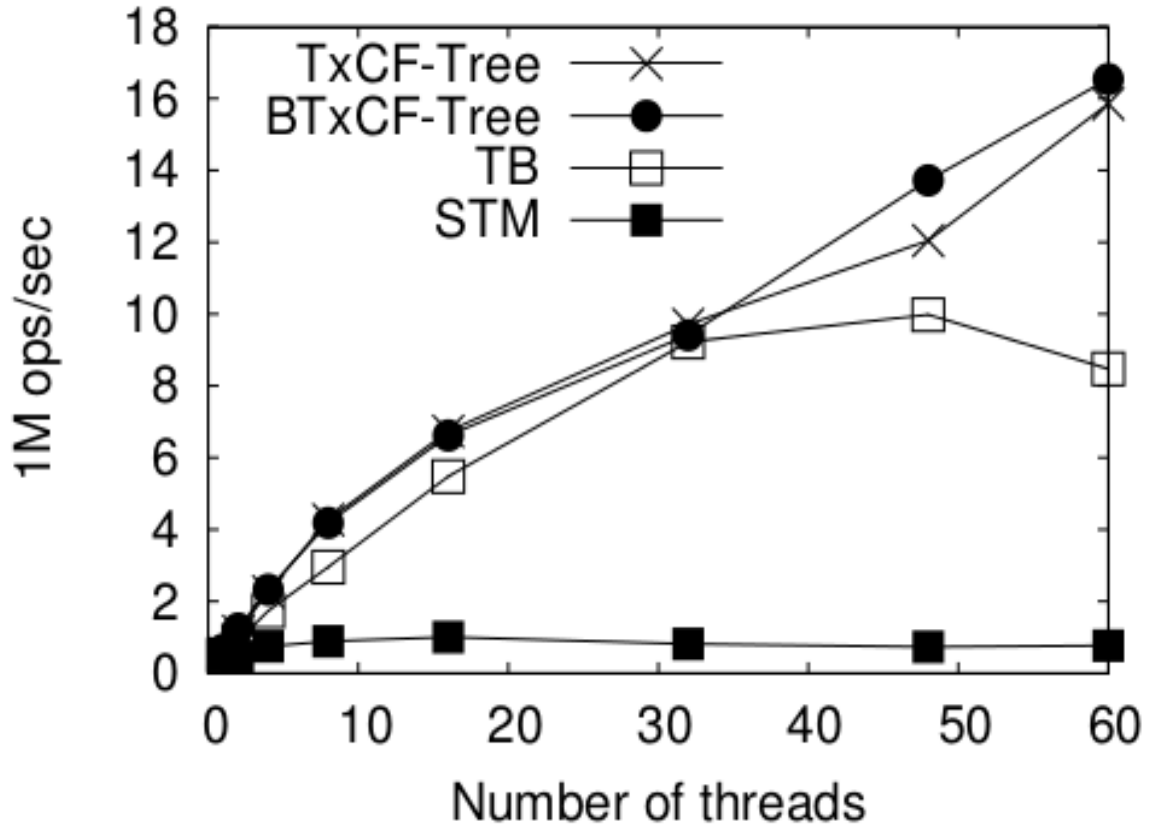
- The helper thread repeatedly calls a recursive depth-first procedure to traverse the entire tree.

Adaptive Back-off Delay

- The helper thread repeatedly calls a recursive depth-first procedure to traverse the entire tree.

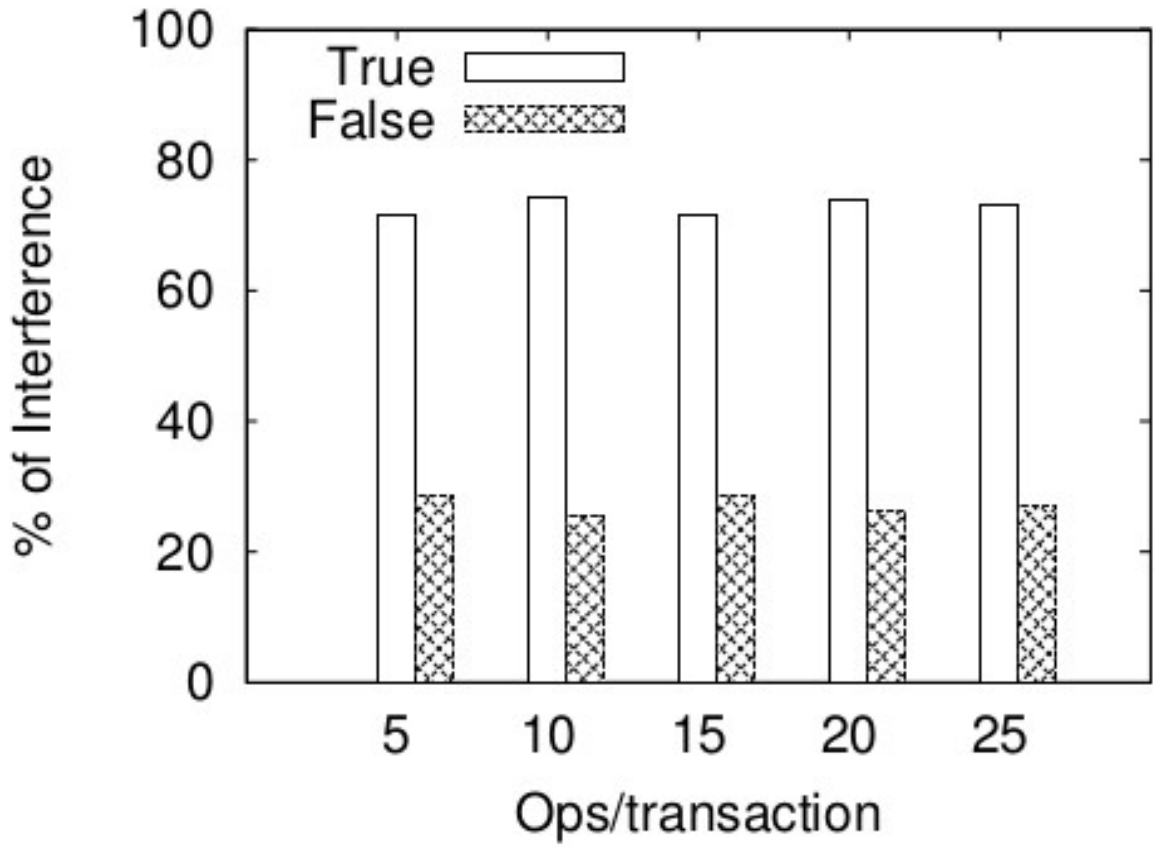
- Our Proposal:
  - Back-off delay between two iterations.
  - Hill-climbing mechanism to increase/decrease the delay.
  - Our Metric: number of housekeeping operations in each iteration.

Evaluation



AMD 64-cores, 10K, 50% reads, 5 ops/transaction

Evaluation



AMD 64-cores, 10K, 32 threads, 50% reads, 5 ops/transaction

Other Trees?

- Hand-over-hand locking
  - Too much overhead.

- Lock-free
  - How to efficiently compose CAS operations.
  - Replace it with lock-based + contention manager.

Other Trees?

- Hand-over-hand locking
  - Too much overhead.
- Lock-free
  - How to efficiently compose CAS operations.
  - Replace it with lock-based + contention manager.
- “Transactional Acceleration of Concurrent Data structures” (SPAA'15).

Conclusion

- Concurrent Balanced Trees are well-designed and optimized to reduce the effect of re-balancing

- TxCF-Tree
  - Boost the functionality to support the composition of operations
  - Reduce the interference of the structural operations (e.g. rotations and physical deletions.
  - Generality Vs Optimization trade-off.

Thanks!

Questions?