Concurrent Augmented Trees*

PANAGIOTA FATOUROU

Foundation for Research and Technology – Hellas, Institute of Computer Science

University of Crete, Department of Computer Science, Greece

Joint work with ERIC RUPPERT, York University, Canada

DISC 2024 (Best Paper Award)

ApPLIED, June 2025

* Supported by the Hellenic Foundation for Research and Innovation (HFRI) under the "Second Call for HFRI Research Projects to support Faculty Members and Researchers" (project number: 3684).

Main Result

Technique to augment lock-free search trees in order to support more operations.

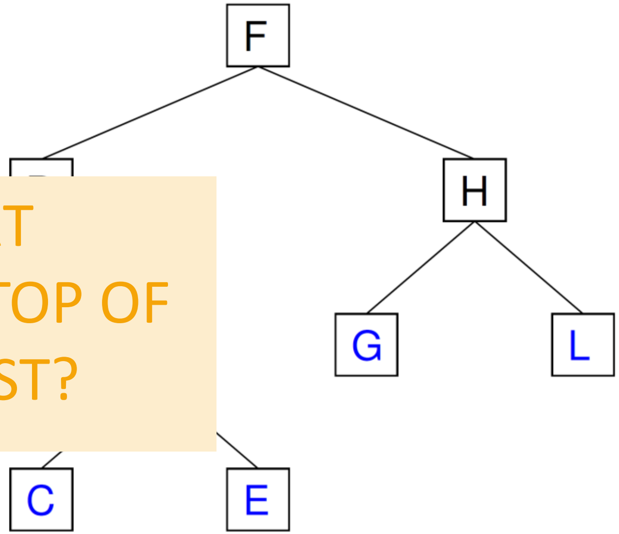
- Simple to implement using single-word CAS
- General: can handle any augmentation
- Efficient: queries as fast as in sequential system,
- minimal overhead for updates
- Wait-free: additional work for augmentation is wait-free
- Snapshots of tree easily support complex queries

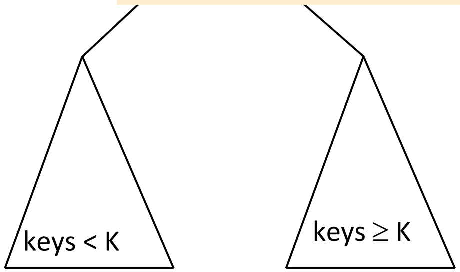
Leaf-Oriented BST

Properties

- One leaf for each key in set
- Internal nodes used for routing
- The tree is full
- BST Property

How TO SUPPORT AUGMENTATION ON TOP OF A CONCURRENT BST?



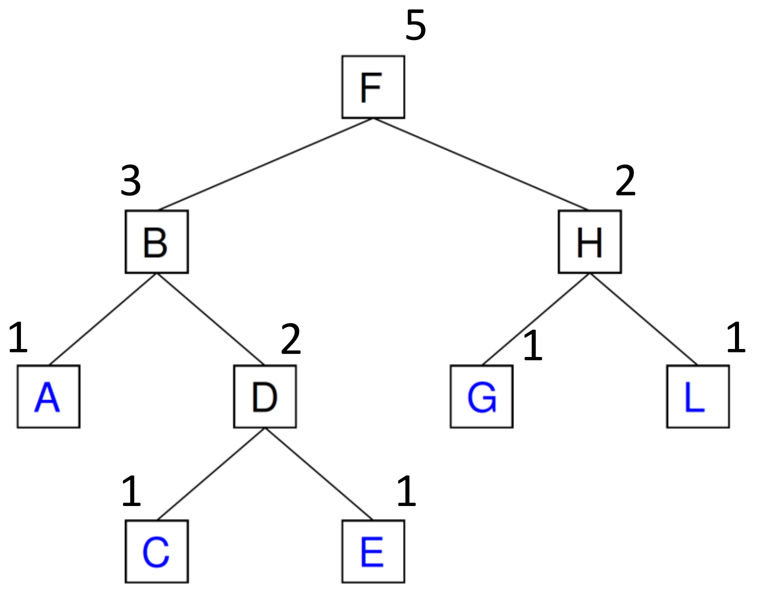


Set = {A, C, E, G, L}.

AUGMENTATION

WHAT CAN WE DO WITH AN AUGMENTED TREE?

Augmentation: Each node stores the number of leaves in its subtree.



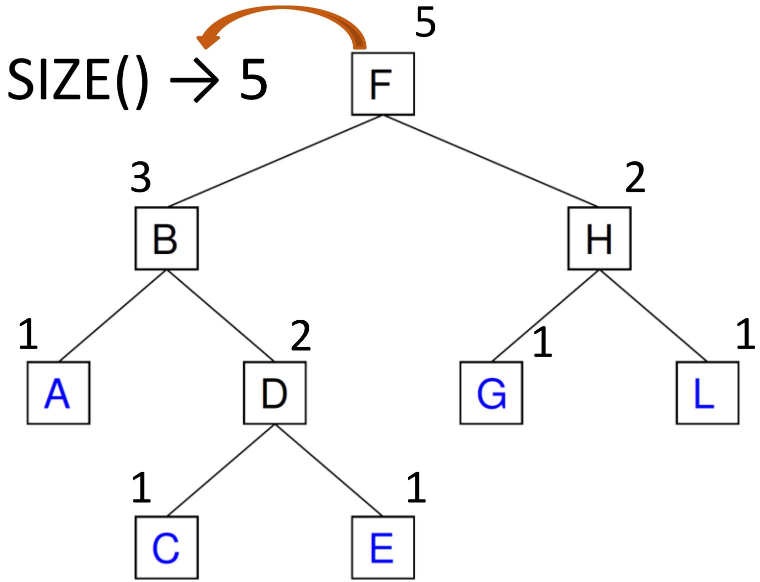
Order-statistic tree

AUGMENTATION

WHAT KIND OF FUNCTIONALITY DO WE WANT TO SUPPORT?

SIZE(): Returns the size of the implemented set.

- O(1) time



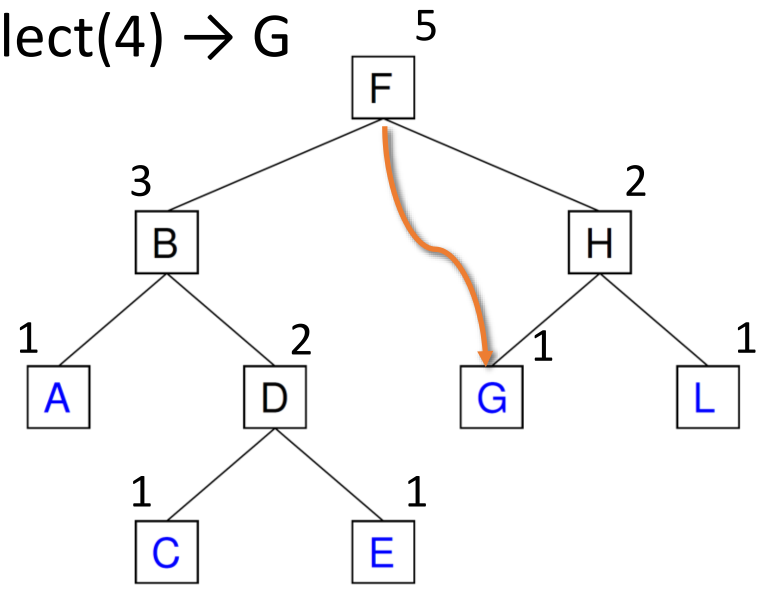
AUGMENTATION

WHAT KIND OF FUNCTIONALITY DO WE WANT TO SUPPORT?

SELECT(i): Returns the i-th largest element in the set.

➢ O(h) time

Select(4) → G

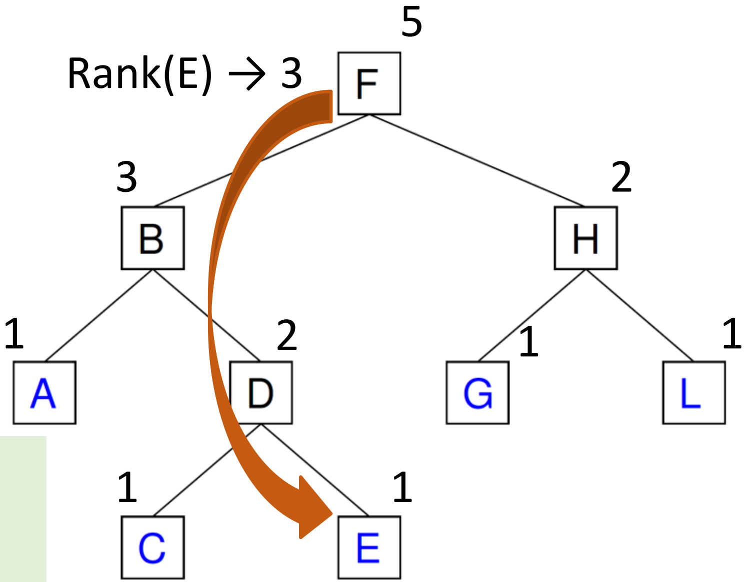


AUGMENTATION

WHAT KIND OF FUNCTIONALITY DO WE WANT TO SUPPORT?

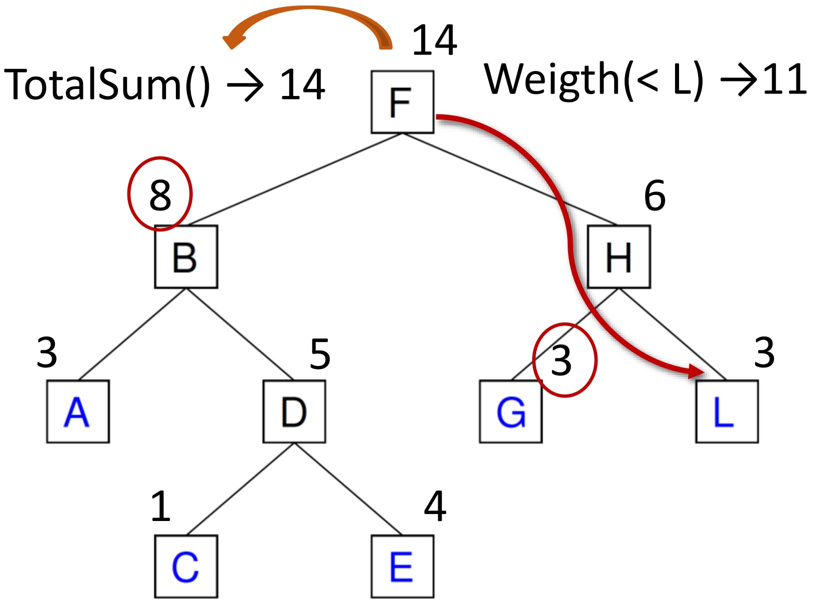
RANK(x): Returns i if x is the i-th largest element in the set.

- O(h) time



EXAMPLES OF AUGMENTATION

- Any associative aggregation operator
  - sum, minimum, maximum, product, etc.
- Augment the tree to filter values
  - obtain the aggregate of all odd values within a range.
- Interval tree
  - Stores a set of intervals in a balanced BST sorted by the left endpoints
  - Each node stores the maximum right endpoint of any interval in the node's subtree
  - Determine whether any interval in the BST includes a given point in logarithmic time



Keys have weights; each node stores sum of subtree's weights.

EXAMPLES OF AUGMENTATION

Many other ways to augment a BST.

- For database of employees, number of women in subtree.
- How many employees' salaries are more than 100,000 euros/year?
- Store min key, max key, and smallest gap in subtree.
- Find two closest keys in the set.

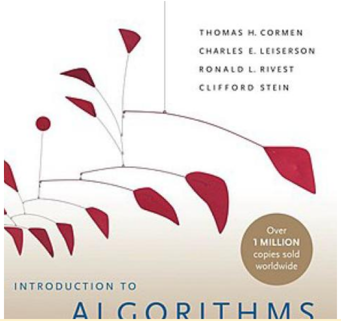
Key Property of Augmentations

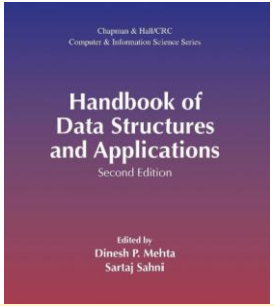
Values of a node's new field(s) can be computed from information in the node and its children.

APPLICATIONS OF AUGMENTATION

Augmented BSTs are basis of building many other data structures.

- Interval tree
- Tango tree
- Measure tree
- Priority search tree
- Segment tree
- Link/cut tree
- several other data structures





Augmentation is sufficiently important to warrant a chapter in classical algorithms textbooks.

Lots (

- computational geometry
- databases
- graph algorithms
- many other fields

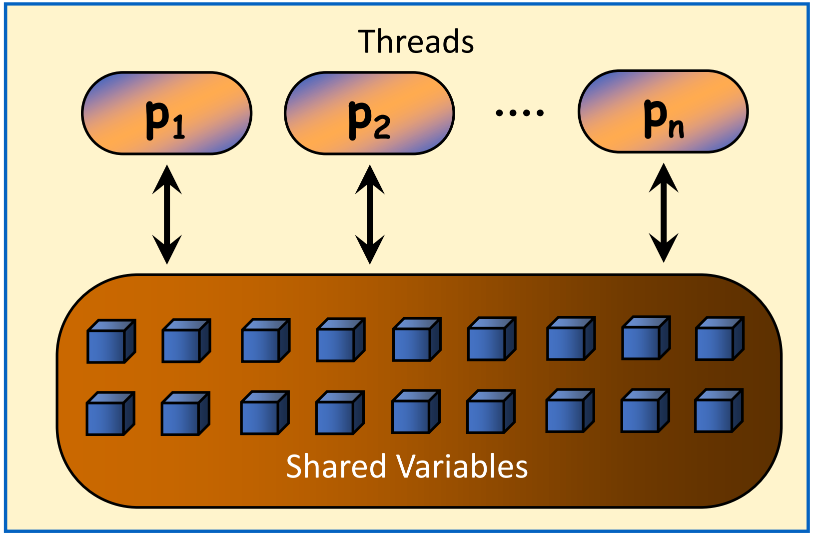
Main Result

Technique to augment concurrent trees

Example: Lock-free Binary Search Tree

- Can handle any augmentation.
- Adds only O(height) steps to insert, delete.
- Supports simple snapshots.
- Wait-free queries run sequential code.
- Based on BST of Ellen et al. from PODC 2014.

Model



- Asynchronous system.
- Communication by accessing shared variables.
- Threads may be delayed indefinitely (or crash).

Correctness - Linearizability

Each operation appears as if it has been executed atomically at some point in its execution interval.

Progress – Lock Freedom

Some thread makes progress

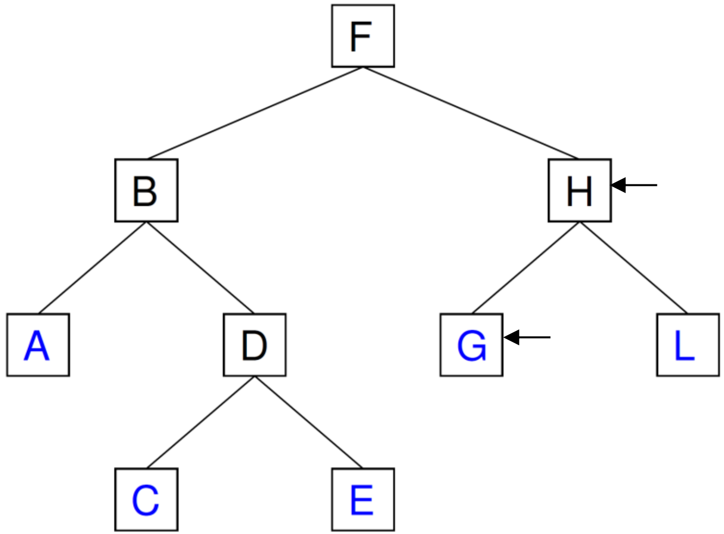
Wait-Freedom

Every non-crashed thread makes progress

Tree Updates – Insert in Leaf-Oriented Tree

Insert(J)

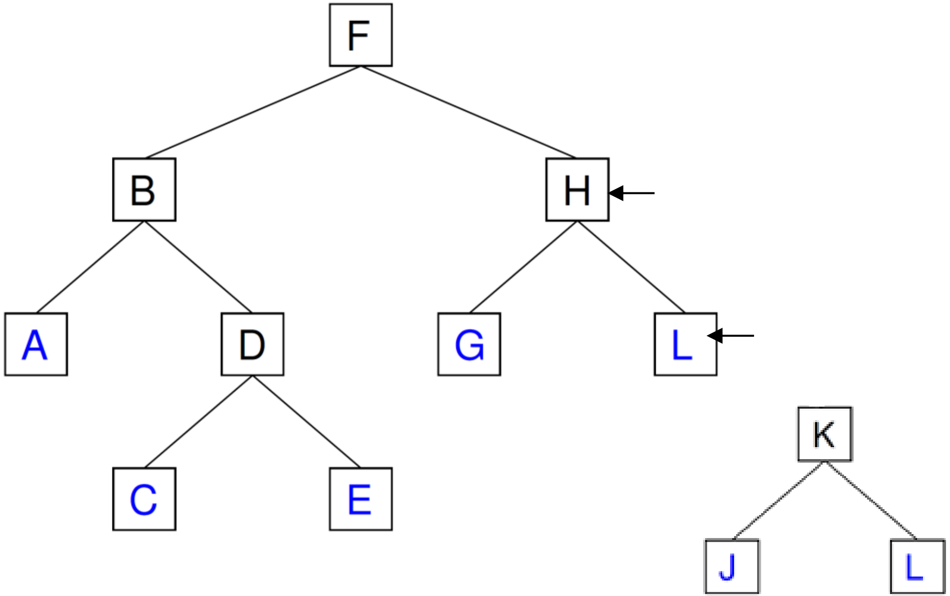
- Search for J
- Remember leaf and its parent



Insert in Leaf-Oriented Tree (Sequentially)

Insert(J)

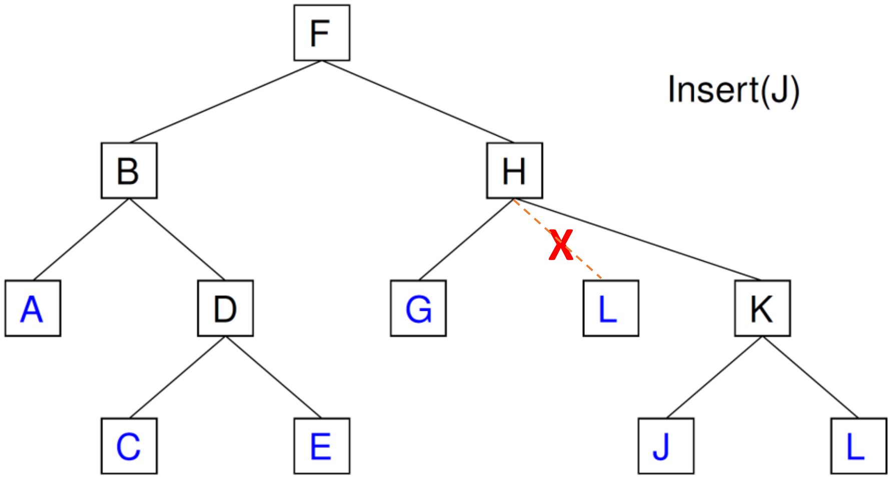
- Search for J
- Remember leaf and its parent
- Create new leaf, replacement leaf and one internal node



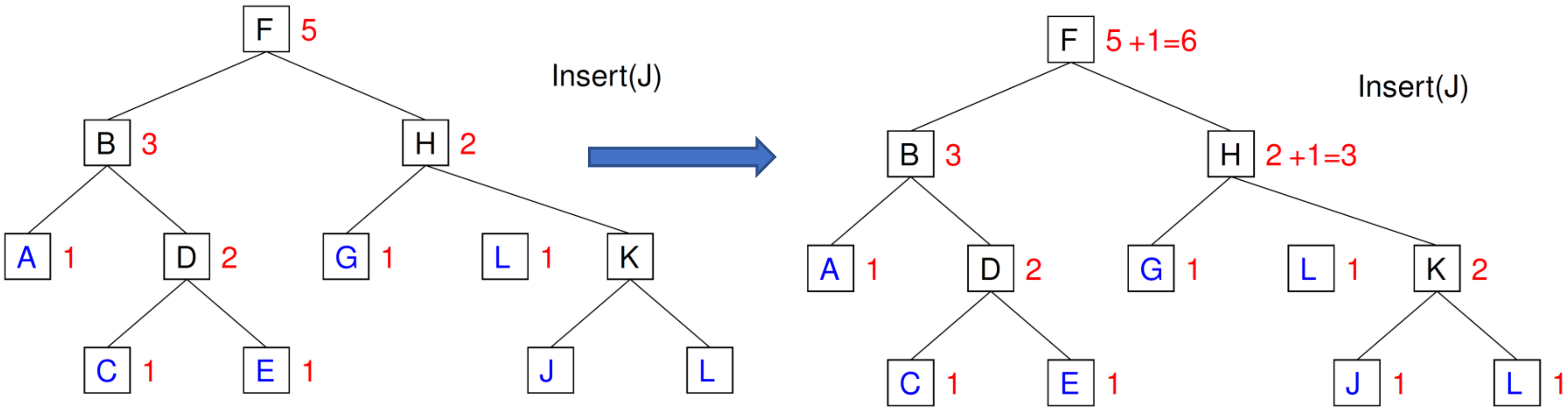
Insert in Leaf-Oriented Tree (Sequentially)

Insert(J)

- Search for J
- Remember leaf and its parent
- Create new leaf, replacement leaf and one internal node
- Update pointer



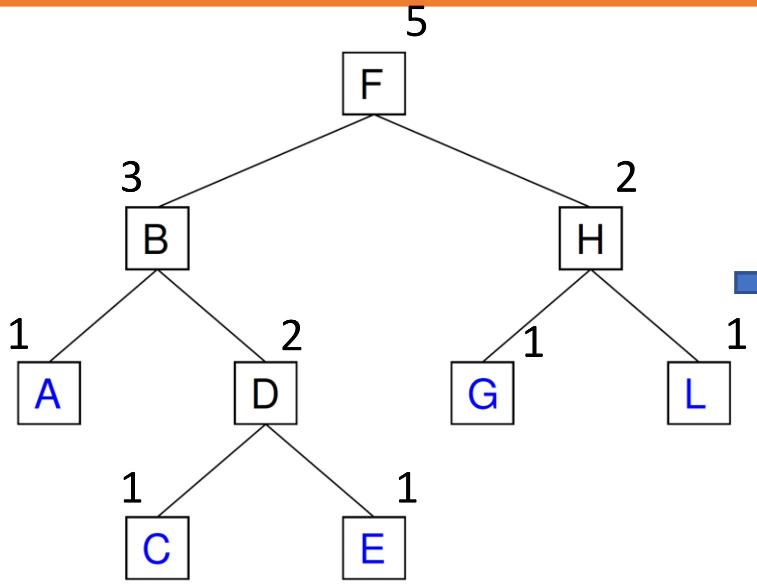
Updates on Augmented Trees

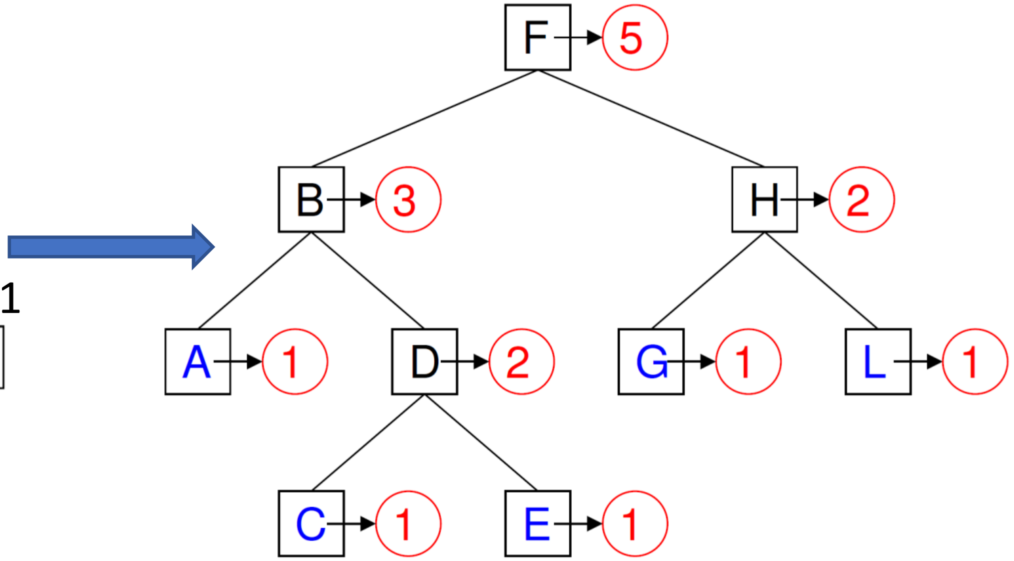


Challenges of Concurrency

- An update changes fields of many nodes along a path
- All changes must appear atomic
- Queries traverse a path while concurrent updates change it
- Contention: all updates need to modify root's size

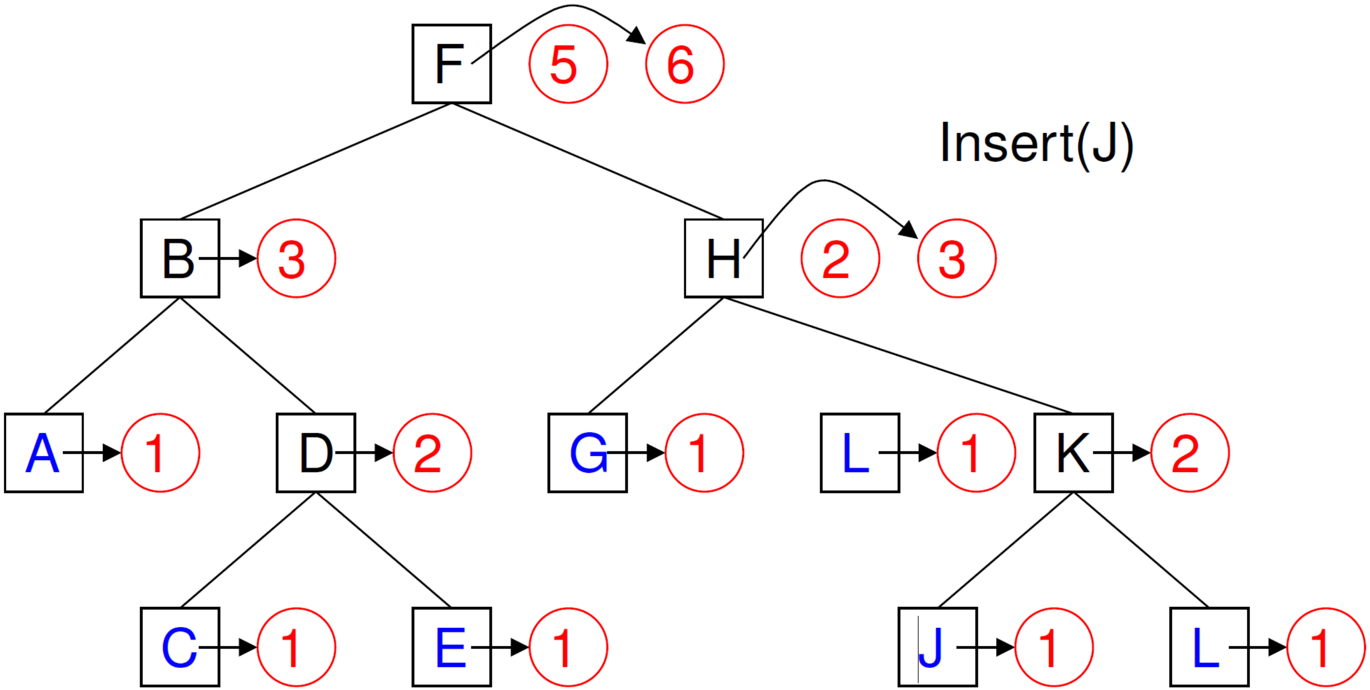
Key Ideas





Node stores pointer to current version of augmented field.

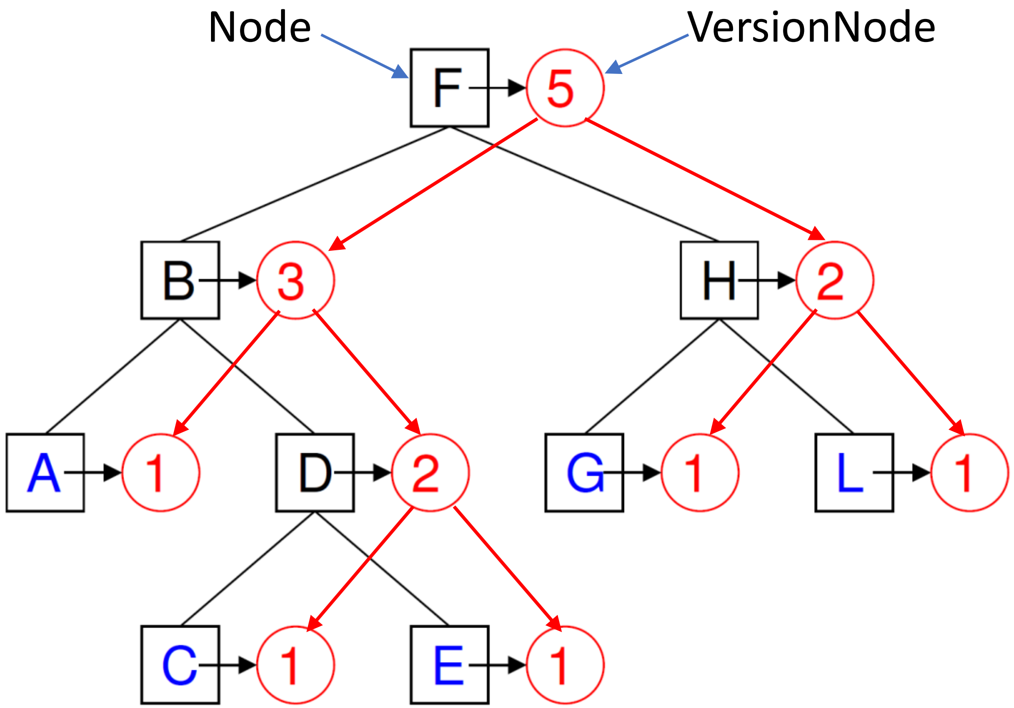
Key Ideas: Multiple Versions of Augmentation fields



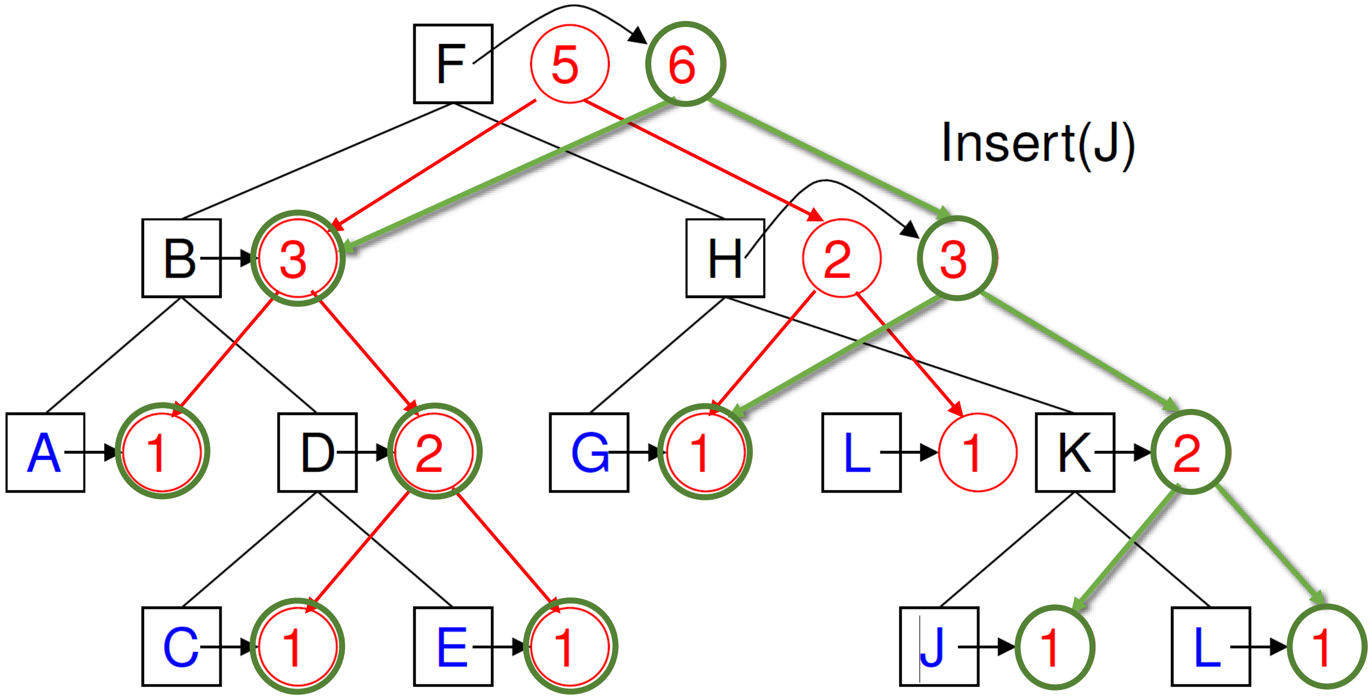
Old versions can still be used by queries in progress

Key Ideas: Version Tree

- Old versions can still be used by queries in progress.
- All fields of VersionNodes are immutable.

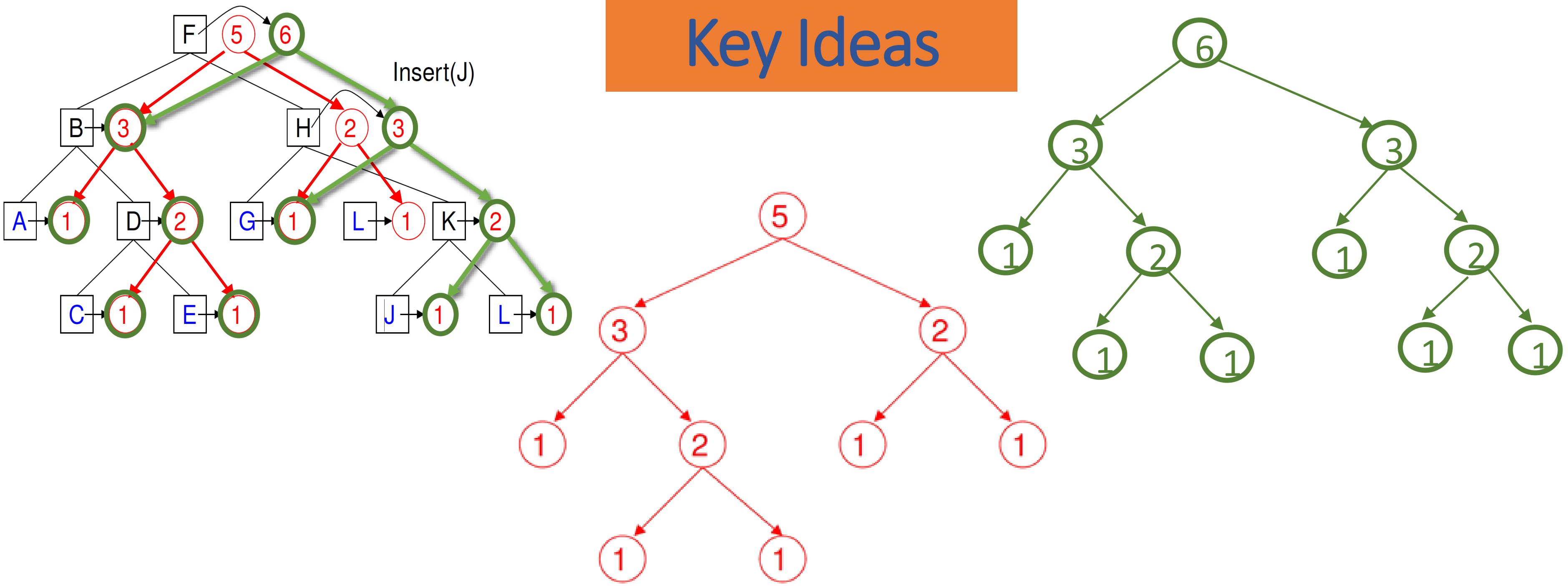


Key Ideas: Multiple Versions of Augmentation fields



VersionTrees provide consistent views

Key Ideas



Accessing root's version provides snapshot of version tree.

Versions also store keys to direct searches.

Supports any sequential query operation.

Old versions are unreachable when no longer needed.

Updating Versions

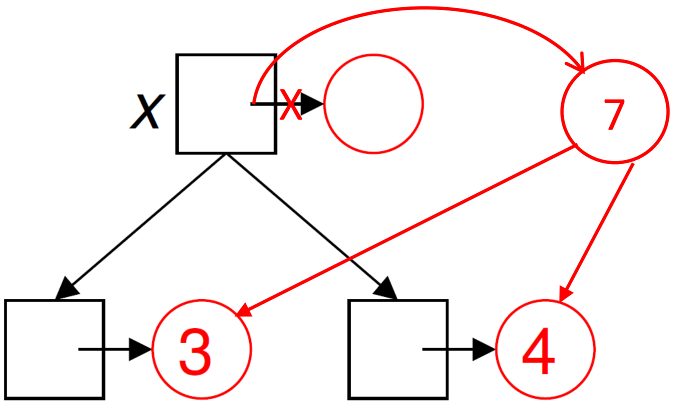
After Insert or Delete, Propagate changes up to root.

Propagate

for each Node x on path to root do (at most) twice

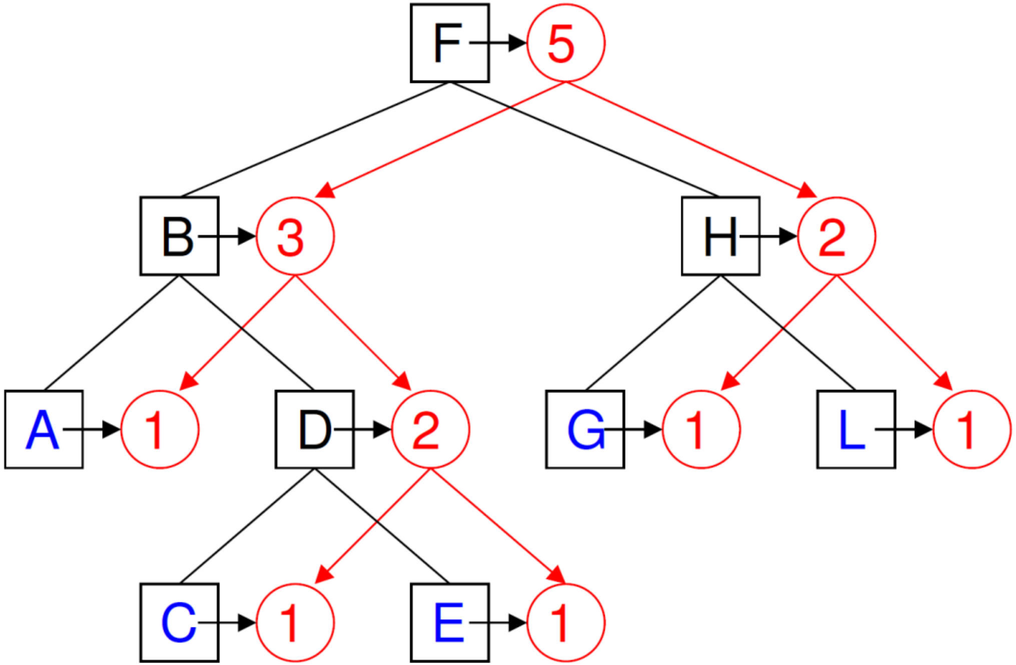
// Refresh x's version

create new VersionNode v
v.left := x.left.version
v.right := x.right.version
compute contents of v
CAS x.version to point to v

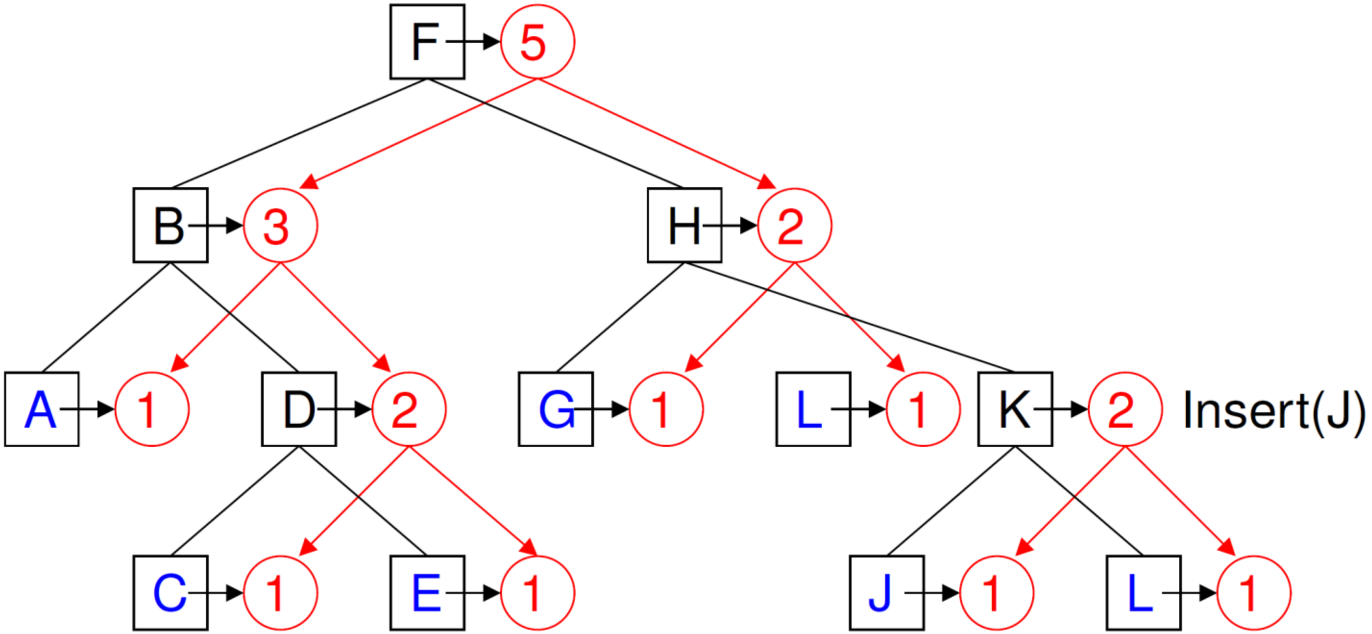


- Fields of VersionNodes never change once it is attached to tree.
- Propagation is wait-free.

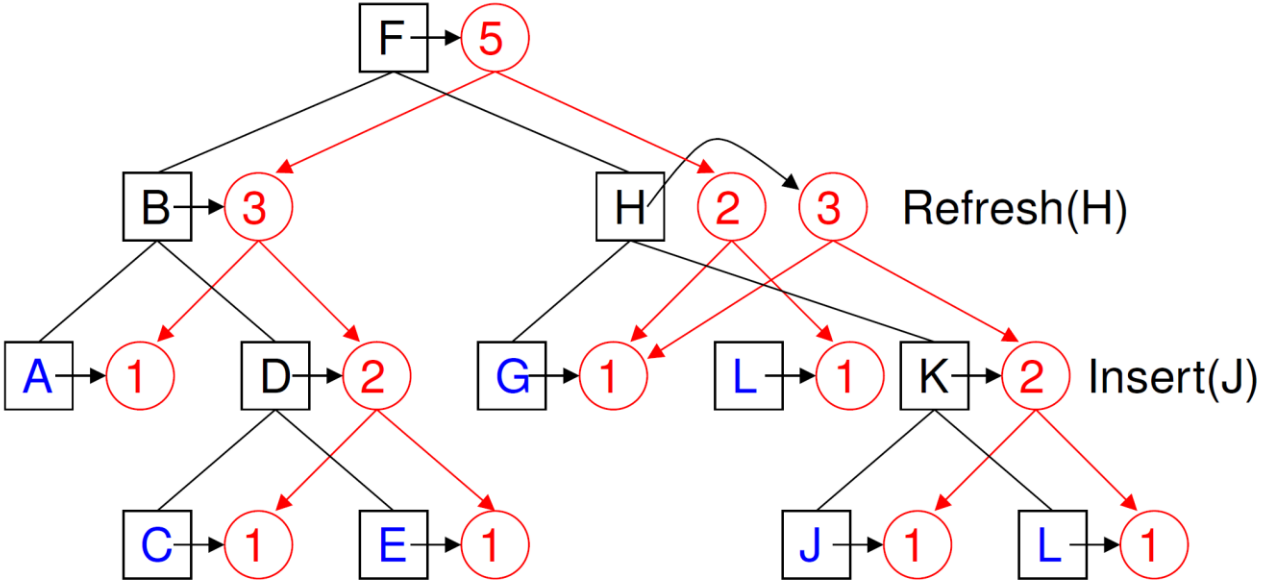
Propagating an Insert



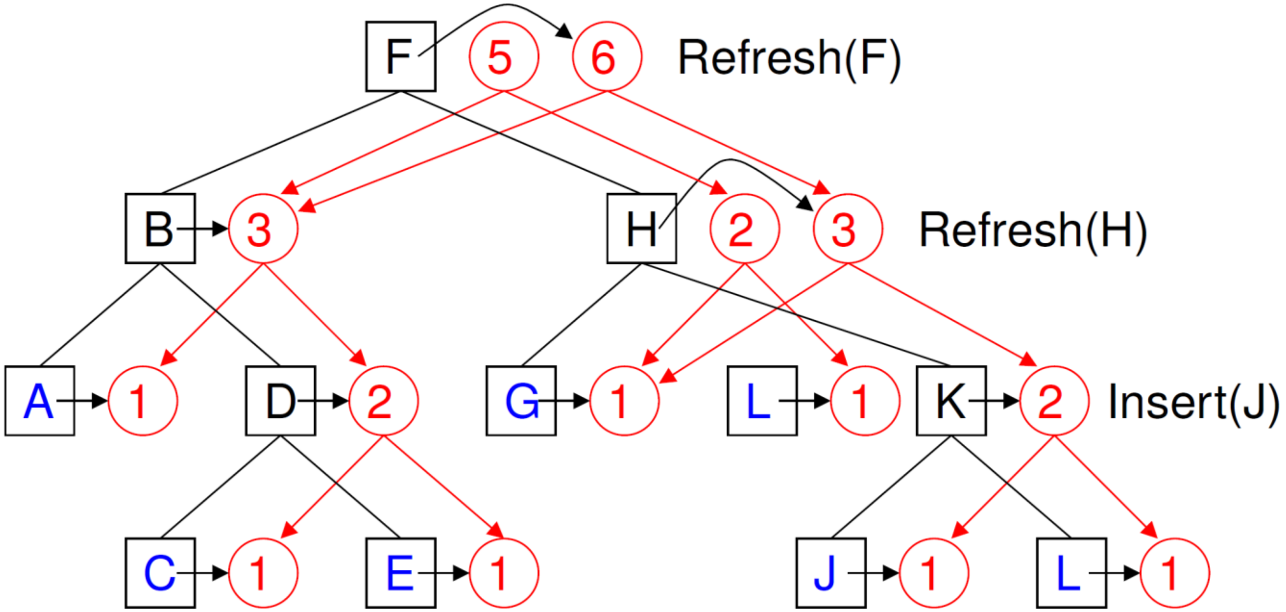
Propagating an Insert



Propagating an Insert



Propagating an Insert



Double Refresh

Refresh on each node x uses CAS to update x's version.

What if the CAS fails?

- Try again

What if the CAS fails again?

- Stop; someone else's refresh has propagated your change to x.

Cooperation and Contention

- Updates are propagated cooperatively
- One change can propagate many operations together
- All update operations perform CAS on root
- BUT not all have to succeed

Algorithm

Insert or Delete operation

- Run original algorithm to perform update
- Refresh each ancestor (at most) twice

➢ Adds O(h) to step complexity of updates.

Query operation

- Read Root.version to get snapshot of version tree
- Run standard sequential algorithm on that snapshot

➢ Step complexity same as sequential query time.

Proving Correctness

Key Goal

- Define linearization points for updates, so that the Version tree rooted at Root.version reflects all updates linearized so far.

➢ Linearize an update when its info has been propagated to Root Node.

We linearize a query at the time it reads Root.version to get a snapshot of the Version tree.

Proving Correctness

Linearizability

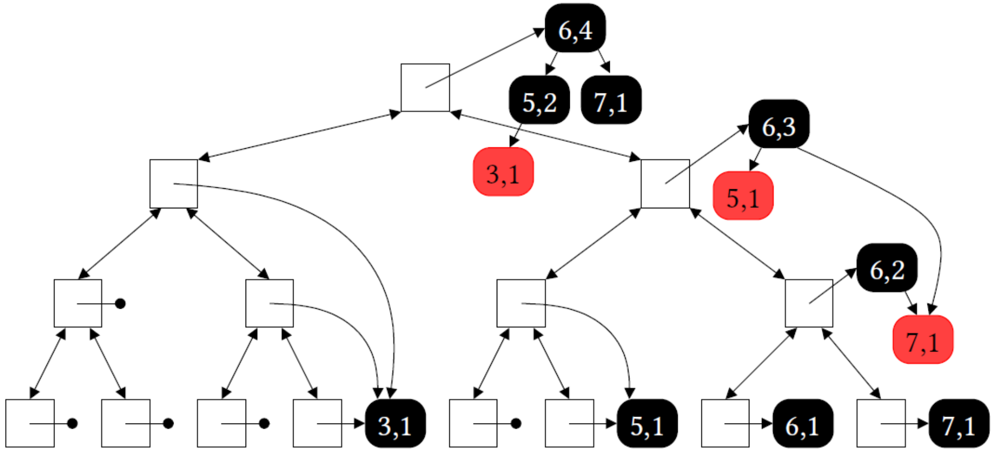
- Define arrival point of each update at each node on its leaf-to-root path.
- Invariant: tree rooted at x.version reflects all operations that have arrived at x (done in order of their arrival points).
- Linearization point = arrival point at Root.
- Show that propagate ensures all operations arrive at Root.

Improving Query Step Complexity

Version of x stores pointer to root of RBT containing all elements in x's subtree.

Refresh:

- Read RBTs pointed to by x.left.version & x.right.version
- Join them into one RBT
- Use CAS to store root of new RBT in x.version



- Let n = |S|.
- Join can be done in O(log n) time.
- Update takes additional O(height log n) steps.
- Queries take O(log n) steps, even if tree height is n.

Comparison to Related Work

Lock-free BST augmented with size

[Kokorin, Alistarh, Aksenov IPDPS 2024]

- Each operation must join a queue at each node and help all those ahead.
- Not generalizable to other augmentations.
- ((#processes) height) steps per operation.

Lock-based tree augmentation

[Sela, Petrank DISC 2024]

- Much work on taking snapshots of shared data structures
- They are more complicated, and have slower queries
- Those based on multiversioning have complex GC

Double refresh has been used in other ways

[Afek, Dauber, Touitou 1995]

Conclusion

Scheme for augmenting concurrent trees

- ✓ is simple to implement
- ✓ works for any augmentation
- ✓ adds O(height) to step complexity of updates
- ✓ preserves lock-freedom or wait-freedom of updates
- ✓ has wait-free, fast queries
- ✓ supports simple snapshots

Thank you!

https://persist-project.gr/

faturu@csd.uoc.gr
www.ics.forth.gr/~faturu/

persist H.F.R.I. Hellenic Foundation for Research & Innovation