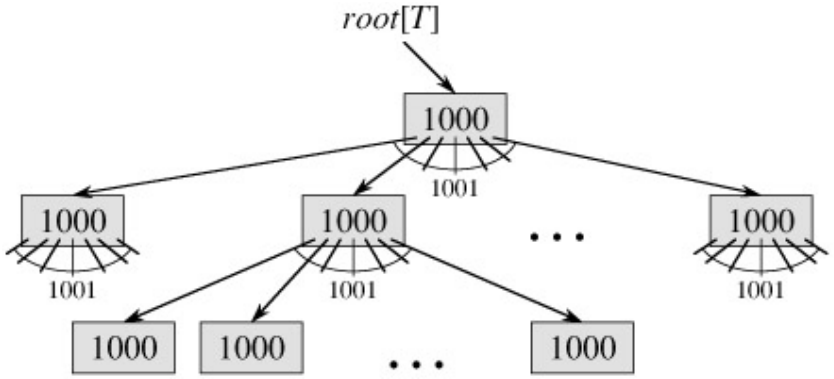
CONCURRENT B-TREE ALGORITHM Implementation

COSC6490A

Liping Han

B-Tree

- Data items are stored in leaf nodes
- Each Internal node include size-1 keys and size downlinks to its children, $\lceil M/2 \rceil \leq size \leq M$
- Within each node keys are in ascending order
- All leaf nodes have same depth



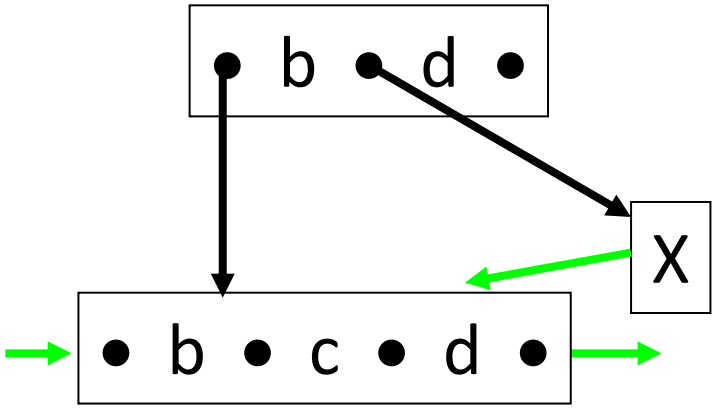
1 node,
1000 keys

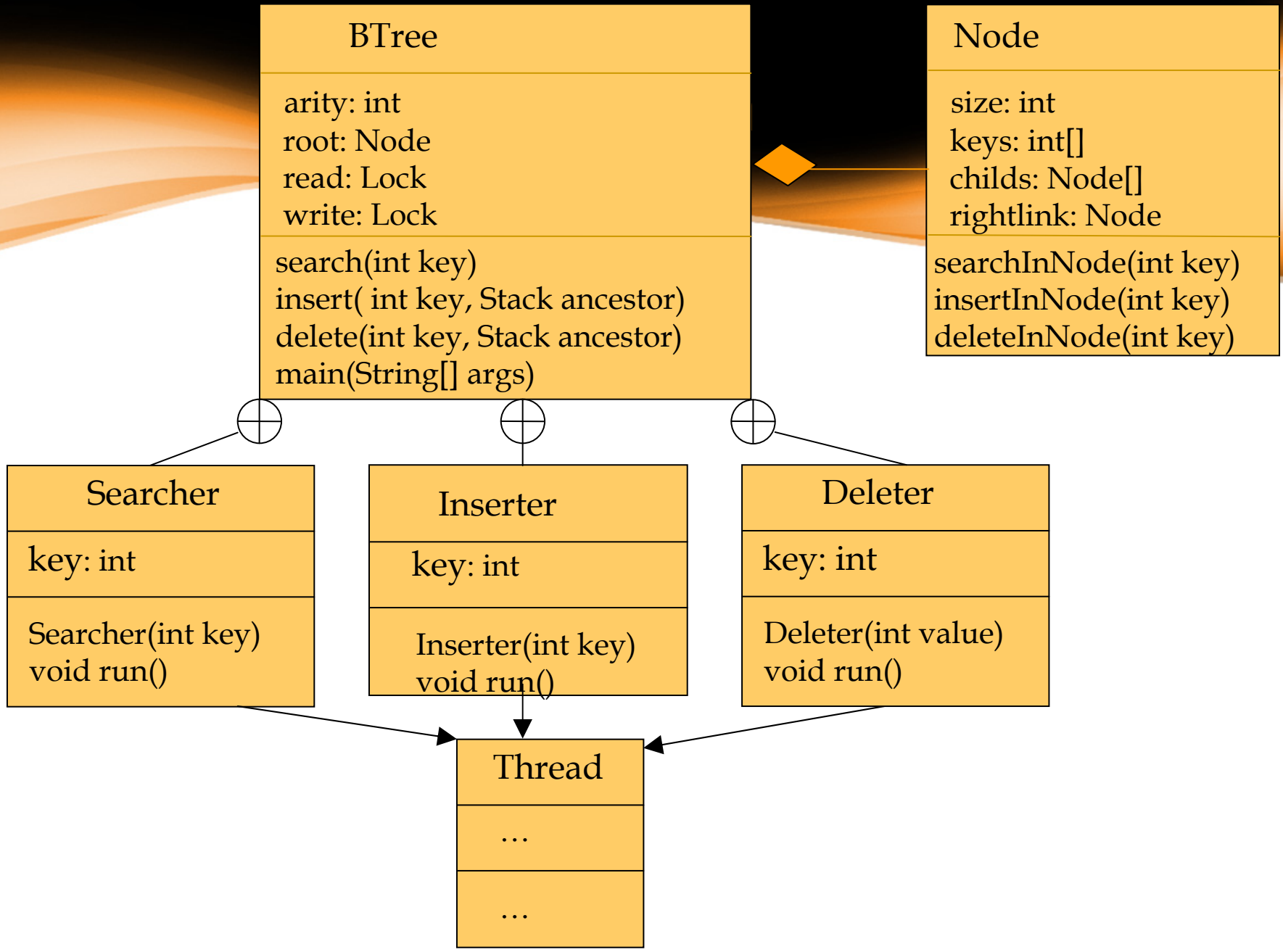
1001 nodes,
1,001,000 keys

1,002,001 nodes,
1,002,001,000 keys

Variants for Concurrent

- Each internal node has a rightlink to the node right to it at the same level
- Empty node add an outlink to the node where its content is merged to




BTree
arity: int
root: Node
read: Lock
write: Lock
search(int key)
insert(int key, Stack ancestor)
delete(int key, Stack ancestor)
main(String[] args)
Node
size: int
keys: int[]
childs: Node[]
rightlink: Node
searchInNode(int key)
insertInNode(int key)
deleteInNode(int key)
Searcher
key: int
Searcher(int key)
void run()
Inserter
key: int
Inserter(int key)
void run()
Deleter
key: int
Deleter(int value)
void run()
Thread
…
…

Lock Granularity

- Lock on nodes
- General case lock on one node
- Early-lock-releasing

ReaderWriter Solution

- private ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();
- private Lock read = rwl.readLock();
- private Lock write = rwl.writeLock();

Algorithms – Phase 1

- Locate

  …

  read.lock

  findpath

  read.unlock

  …

Algorithms – Phase 2

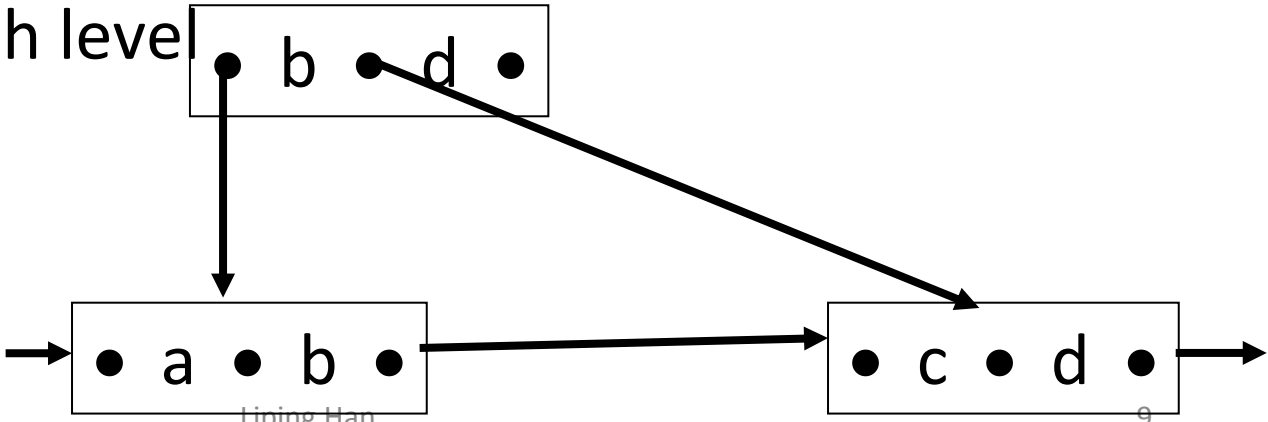
- Action on leaf node
  - Search: read lock
  - Insert: write lock
  - Delete: write lock

Algorithms – Phase 3

- Normalization: two phase split and merge
  - Half-split/half-merge: not involve parent
  - Add-link/remove-link: take place on node of level above half-split/half-merge

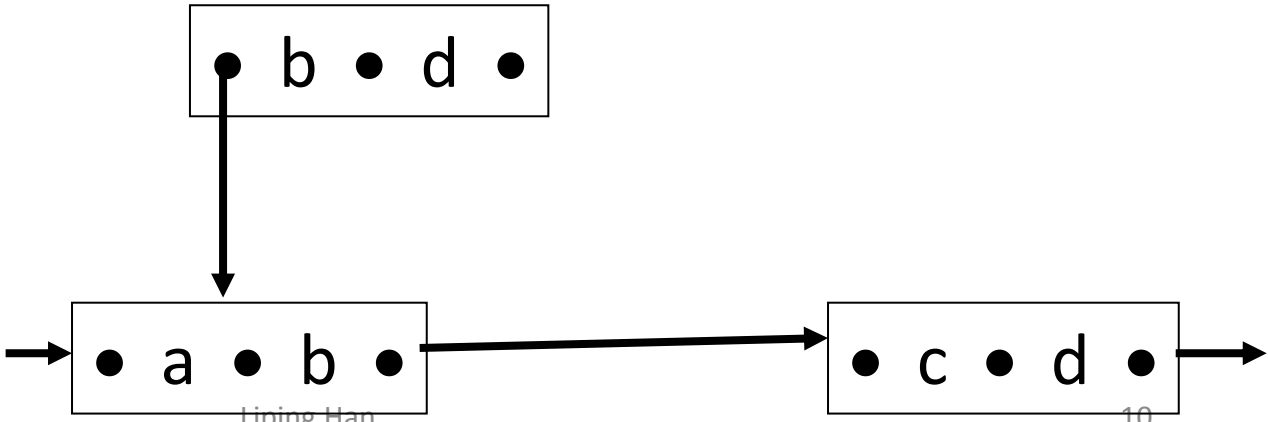
Extra link: extra work

- During locate phase, traversal may follow rightlink or outlink instead of downlink
  - At each level, scan node by moving right (or follow outlink if node is empty) to find the exact node where the key falls in its coverset
  - Record the node and the first key bigger than target at each level

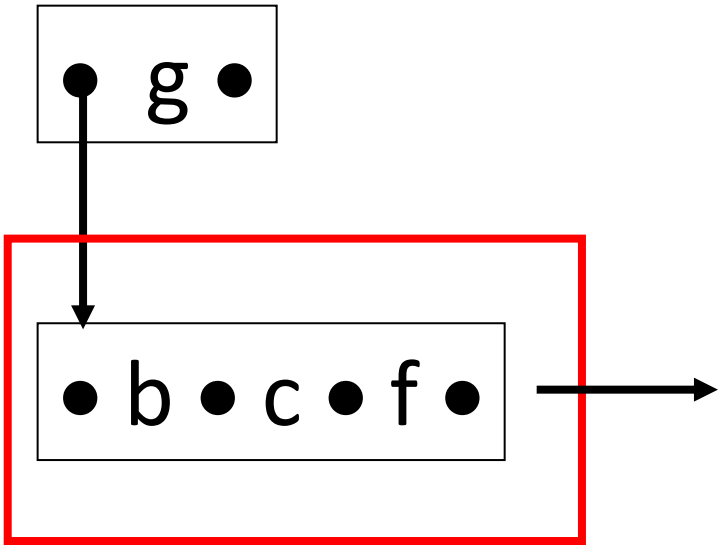


Extra link: extra work

- During the ascendants of normalization, a node may have several or no parent
  - Let s be the rightmost key in the left node after a half-split or before a half-merge
  - The parent node should be the one where s falls in its coverset

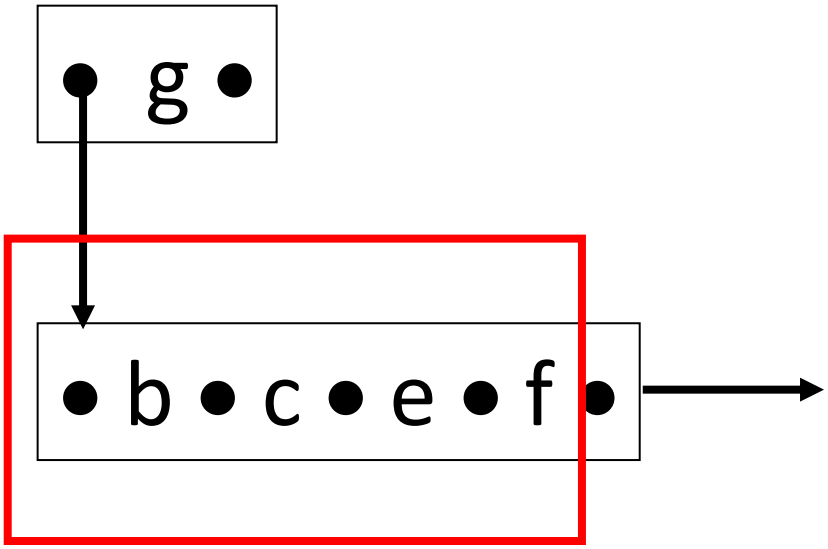


Multi-nodes to lock



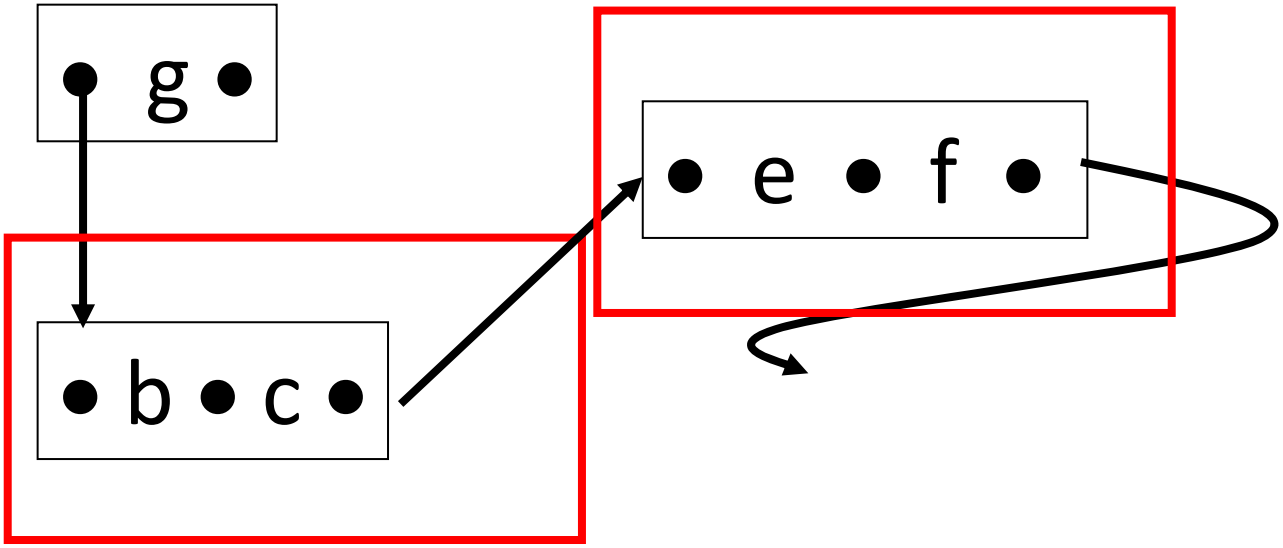
Insert(e)

Multi-nodes to lock



Insert(e)

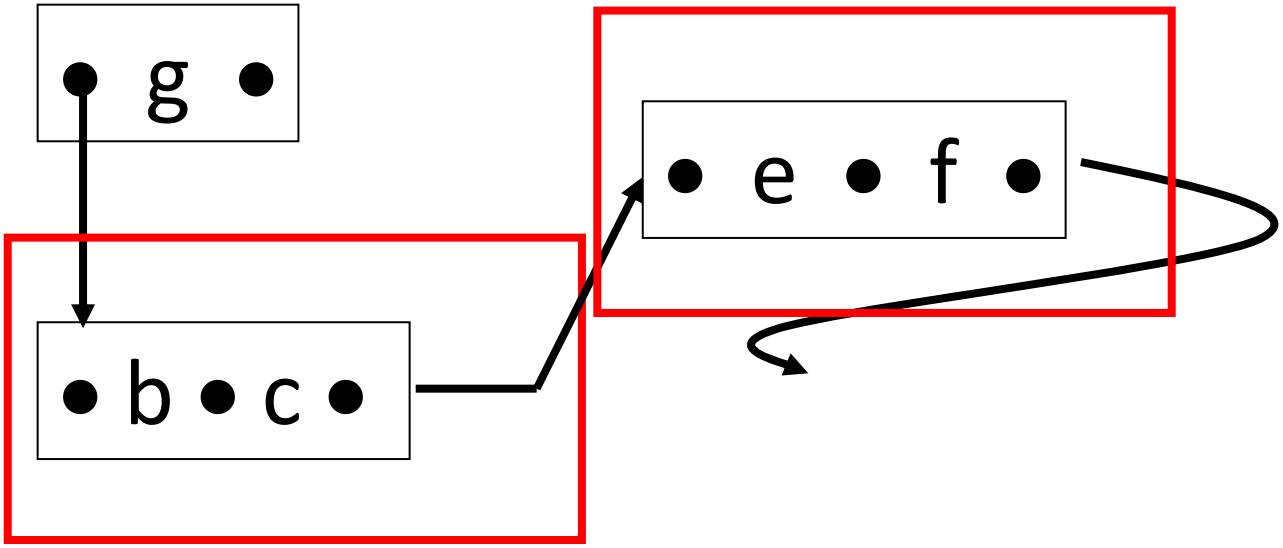
Multi-nodes to lock



Insert(e)

Normalization: half-split

Multi-nodes to lock



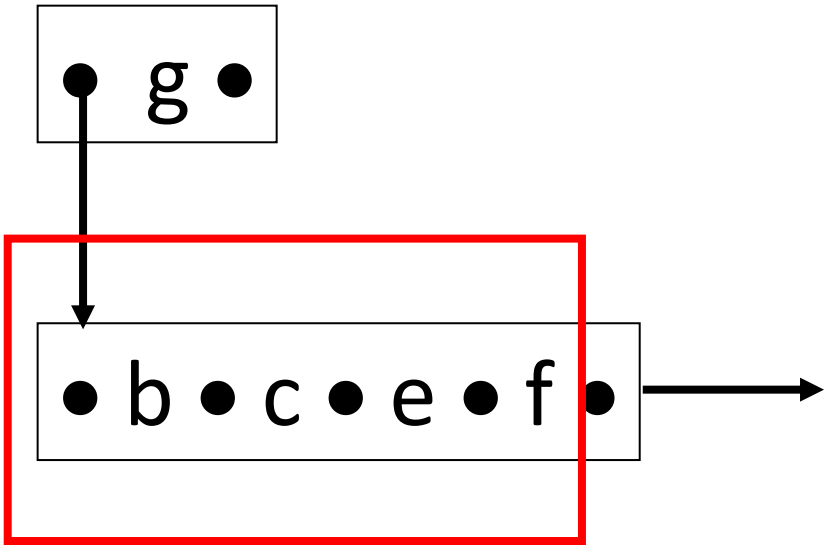
Insert(e)

↓

Normalization: half-split

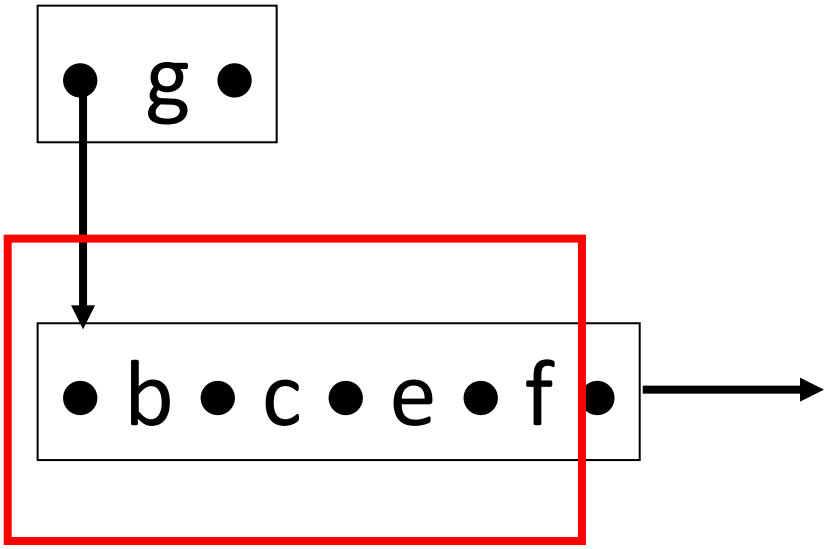
Delete(c): - wait

Multi-nodes to lock



Insert(e) → Insert(d)

Dead-lock?



Insert(e) → Insert(d)

Normalization: half-split

Next Step

- Experimental on algorithm performance
- Try ascendant approach

Papers

- Lehman, P. L. and Yao, s. B. 1981. Efficient locking for concurrent operations on B-trees. *ACM Trans. Database Syst.* 6, 4 (Dec. 1981), 650-670. DOI= http://doi.acm.org.ezproxy.library.yorku.ca/10.1145/319628.319663
- Lanin, V. and Shasha, D. 1986. A symmetric concurrent B-tree algorithm. In *Proceedings of 1986 ACM Fall Joint Computer Conference* (Dallas, Texas, United States). IEEE Computer Society Press, Los Alamitos, CA, 380-389.