Index Locking and Concurrency Control

Kathleen Durant PhD
Northeastern University
Lecture 16b

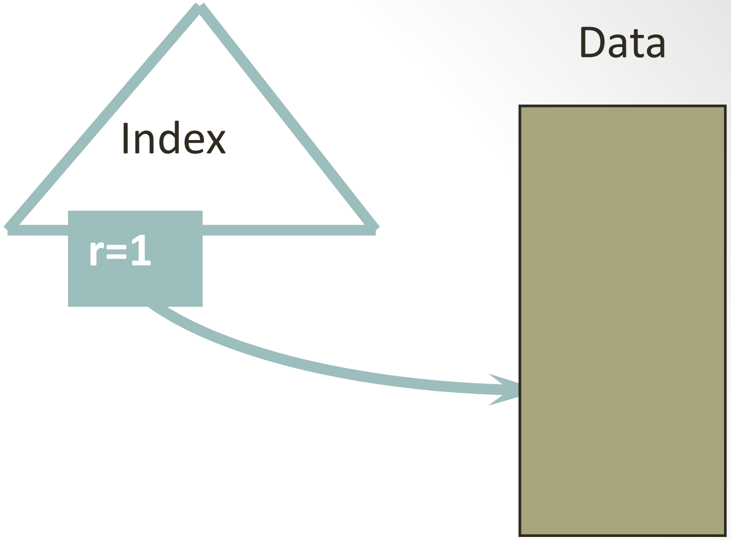
Dynamic Databases

- If we relax the assumption that the DB is a fixed collection of objects, even Strict 2PL will not assure serializability:
  - T1 locks all pages containing sailor records with *rating* = 1, and finds oldest sailor (say, *age* = 71) (Page X)
  - Next, T2 inserts a new sailor; *rating* = 1, *age* = 96.
    - Since the new record can live on a different page – T1 will not have a lock on the page where the new record is inserted (Page Y)
  - T2 also deletes oldest sailor with rating = 2 (and, say, *age* = 80), and commits. (Page Z)
  - T1 now locks all pages containing sailor records with *rating* = 2, and finds oldest (say, *age* = 63).
- No consistent DB state where T1 is "correct"
  - <T1,T2> = <AGES: 71, 80>. <T2,T1> = <AGES: 96,63>
  - This scenario <AGES: 71,63>

The Problem

- T1 implicitly assumes that it has locked the set of all sailor records with *rating* = 1.
  - Assumption only holds if no sailor records are added while T1 is executing
  - Need some mechanism to enforce this assumption. (Index locking and predicate locking.)
- Example shows that conflict serializability guarantees serializability only if the set of objects is fixed

Index Locking



- If there is a dense index on the *rating* field using Alternative (2), T1 should lock the index page containing the data entries with *rating* = 1.
  - If there are no records with *rating* = 1, T1 must lock the index page where such a data entry *would* be, if it existed
- If there is no suitable index, T1 must lock all pages, and lock the file/table to prevent new pages from being added, to ensure that no new records with *rating* = 1 are added.

Predicate Locking

- Grant lock on all records that satisfy some logical predicate, e.g. *age > 2*salary*.
- Index locking is a special case of predicate locking for which an index supports efficient implementation of the predicate lock.
- In general, predicate locking has a lot of locking overhead.

Locking in B+ Trees

- How can we efficiently lock a particular leaf node?
  - Btw, don't confuse this with multiple granularity locking
- One solution: Ignore the tree structure, just lock pages while traversing the tree, following 2PL.
- This has terrible performance!
  - Root node (and many higher level nodes) become bottlenecks because every tree access begins at the root.

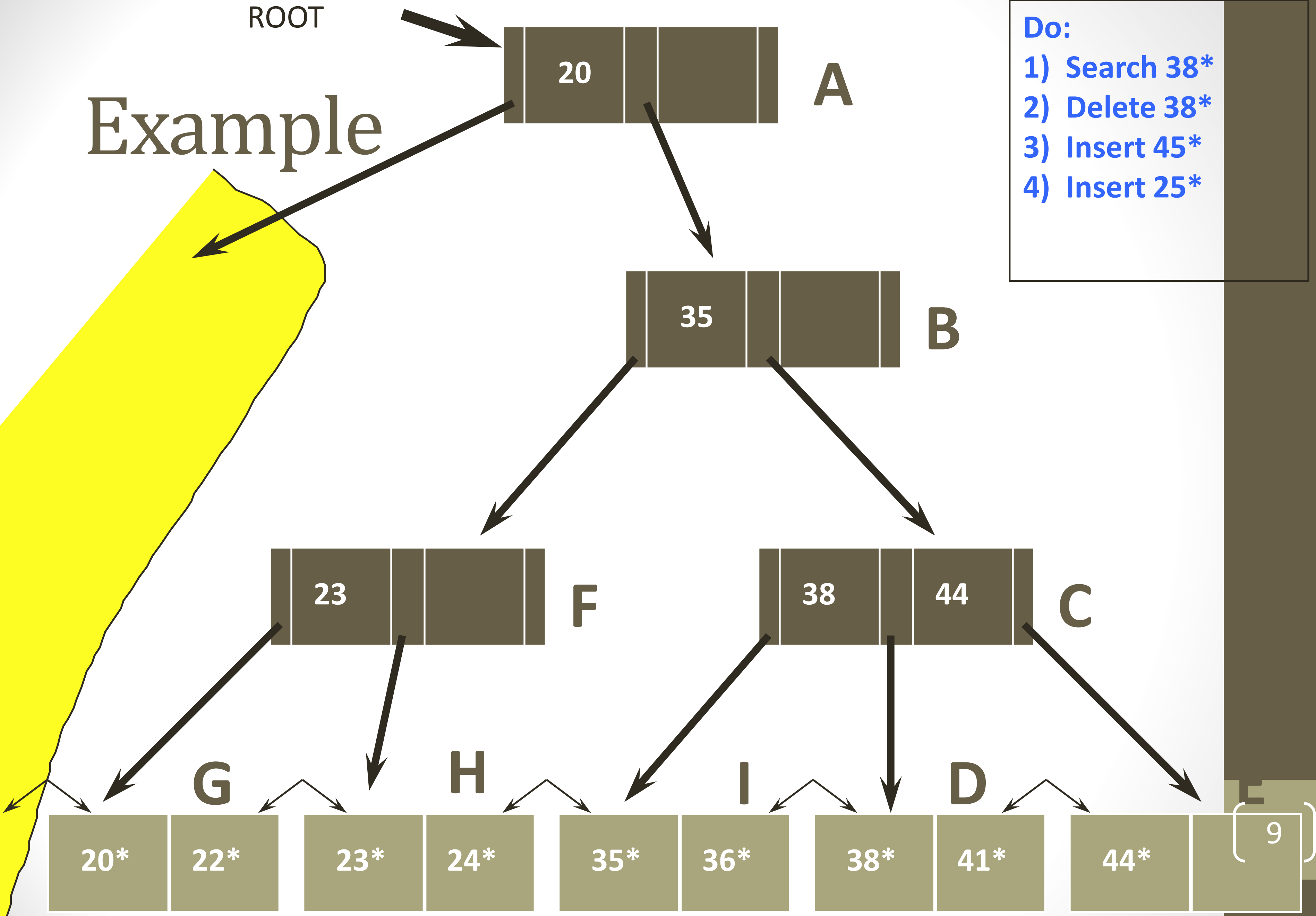
Two Useful Observations

- Higher levels of the tree only direct searches for leaf pages.
- For inserts, a node on a path from root to modified leaf must be locked (in X mode, of course), only if a split can propagate up to it from the modified leaf. (Similar point holds for deletes.)
- We can exploit these observations to design efficient locking protocols that guarantee serializability *even though they violate 2PL.*

A Simple Tree Locking Algorithm

- Search: Start at root and go down; repeatedly, S lock child then unlock parent.
- Insert/Delete: Start at root and go down, obtaining X locks as needed. Once child is locked, check if it is safe:
  - If child is safe, release all locks on ancestors.
- Safe node: Node such that changes will not propagate up beyond this node.
  - Inserts: Node is not full.
  - Deletes: Node is not half-empty.

Example



Do:

1) Search 38*
2) Delete 38*
3) Insert 45*
4) Insert 25*

A Better Tree Locking Algorithm (See Bayer-Schkolnick paper)

- Search: As before.
- Insert/Delete:
  - Set locks as if for search, get to leaf, and set X lock on leaf.
  - If leaf is not safe, release all locks, and restart Xact using previous Insert/Delete protocol.
- Gambles that only leaf node will be modified; if not, S locks set on the first pass to leaf are wasteful. In practice, better than previous algorithm

Summary

- Index locking is common, and affects performance significantly.
  - Needed when accessing records via index.
  - Needed for locking logical sets of records (index locking/predicate locking).
- Tree-structured indexes:
  - Straightforward use of 2PL very inefficient.
  - Bayer-Schkolnick illustrates potential for improvement.
- In practice, better techniques now known; do record-level, rather than page-level locking.