CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

Lecture #22: Concurrency Control
Part 2 (R&G ch. 17)

CMU SCS

Outline

- ✓ conflict/view serializability
- ✓ Two-phase locking (2PL); strict 2PL (== 2PL-C, for 'Commit')
- ✓ deadlocks prevention & detection
- Locking granularity
- Tree locking protocols
- Phantoms & predicate locking

CMU SCS

Review questions

- conflict serializability?
- 2PL theorem?
- what is strict 2PL? why do we need it?
  - 'dirty read'?
  - cascading aborts?
- who generates the lock requests?

CMU SCS

Not in book: 'Lost update' problem

time ↓	T1	T2
	Read(N)	
		Read(N)
	N=N-1	
		N= N-1
	Write(N)	
		Write(N)

CMU SCS

Major conclusions so far:

- **(strict) 2PL: extremely popular**
- Deadlock may still happen
  - detection: wait-for graph
  - prevention: abort some xacts, defensively
- philosophically: concurrency control uses:
  - locks
  - and aborts

CMU SCS

Outline

- ✓ conflict/view serializability
- ✓ Two-phase locking (2PL); strict 2PL (== 2PL-C, for 'Commit')
- ✓ deadlocks prevention & detection
- ➡ Locking granularity
- Tree locking protocols
- Phantoms & predicate locking

CMU SCS

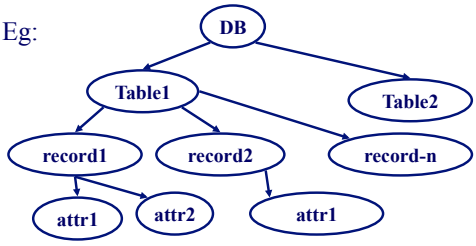
Lock granularity?

- lock granularity
  - field? record? page? table?
- Pros and cons?
- (Ideally, each transaction should obtain a few locks)

CMU SCS

Multiple granularity

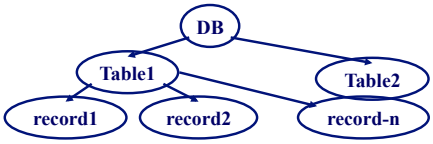
- Eg:



CMU SCS

What would you do?

- T1: read Smith's salary,
- while T2: give 10% raise to everybody
- what locks should they obtain?



CMU SCS

What types of locks?

- X/S locks for leaf level +
- '**intent**' locks, for higher levels

CMU SCS

What types of locks?

- X/S locks for leaf level +
- 'intent' locks, for higher levels
- IS: intent to obtain S-lock underneath
- IX: intent .... X-lock ...
- S: shared lock for this level
- X: ex- lock for this level
- SIX: shared lock here; + IX

CMU SCS

Protocol

- each xact obtains appropriate lock at highest level
- proceeds to desirable lower levels

CMU SCS

Compatibility matrix

T2 wants / T1 has	IS	IX	S	SIX	X
IS					
IX					
S					
SIX					
X					

CMU SCS

Compatibility matrix

T2 wants / T1 has	IS	IX	S	SIX	X
IS	**Y**	**Y**	**Y**	**Y**	**N**
IX					
S					
SIX					
X					

CMU SCS

Compatibility matrix

T2 wants / T1 has	IS	IX	S	SIX	X
IS	**Y**	**Y**	**Y**	**Y**	**N**
IX		**Y**	**N**	**N**	**N**
S					
SIX					
X					

CMU SCS

Compatibility matrix

T2 wants / T1 has	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX		Y	N	N	N
S			Y	N	N
SIX					
X					

Faloutsos SCS 15-415/615 #16

CMU SCS

Compatibility matrix

T2 wants / T1 has	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX		Y	N	N	N
S			Y	N	N
SIX				N	N
X					N

Faloutsos SCS 15-415/615 #17

CMU SCS

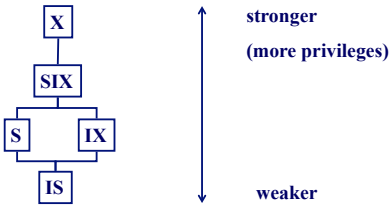
Multiple Granularity Lock Protocol

- Each Xact: lock root.
- To get S or IS lock on a node, must hold **at least** IS on parent node.
  - What if Xact holds SIX on parent? S on parent?
- To get X or IX or SIX on a node, must hold **at least** IX on parent node.
- Must release locks in bottom-up order.

Faloutsos SCS 15-415/615 #18

CMU SCS

Multiple granularity protocol



CMU SCS

Examples – 2 level hierarchy

Tables
Tuples

- T1 scans R, and updates a few tuples:

CMU SCS

Examples – 2 level hierarchy

- T1 scans R, and updates a few tuples:
- T1 gets an SIX lock on R, then get X lock on tuples that are updated.

CMU SCS

Examples – 2 level hierarchy

- T2: find avg salary of 'Sales' employees

CMU SCS

Examples – 2 level hierarchy

- T2: find avg salary of 'Sales' employees
- T2 gets an IS lock on R, and repeatedly gets an S lock on tuples of R.

CMU SCS

Examples – 2 level hierarchy

- T3: sum of salaries of everybody in 'R':

CMU SCS

Examples – 2 level hierarchy

- T3: sum of salaries of everybody in 'R':
- T3 gets an S lock on R.
- OR, T3 could behave like T2; can use lock escalation to decide which.
  - Lock escalation dynamically asks for coarser-grained locks when too many low level locks acquired

CMU SCS

Multiple granularity

- Very useful in practice
- each xact needs only a few locks

CMU SCS

Outline

- ...
- Locking granularity
- Tree locking protocols
- Phantoms & predicate locking

CMU SCS

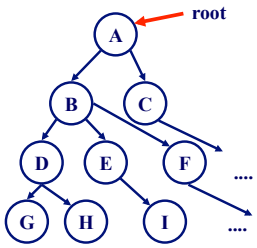
Locking in B+ Trees

- What about locking indexes?

CMU SCS

Example B+tree

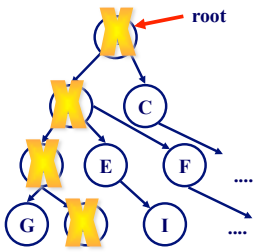
- T1 wants to insert in H
- T2 wants to insert in I
- why not plain 2PL?



CMU SCS

Example B+tree

- T1 wants to insert in H
- T2 wants to insert in I
- why not plain 2PL?
- Because: X/S locks for too long!



CMU SCS

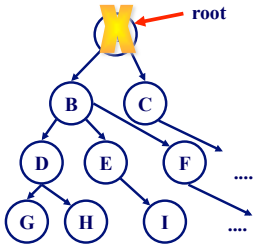
Two main ideas:

- 'crabbing': get lock for parent; get lock for child; release lock for parent (if 'safe')
- 'safe' nodes == nodes that won't split or merge, ie:
  - not full (on insertion)
  - more than half-full (on deletion)

CMU SCS

Example B+tree

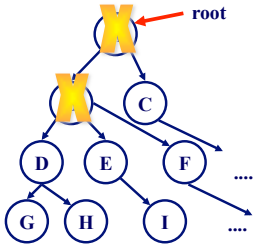
- T1 wants to insert in H
- crabbing:



CMU SCS

Example B+tree

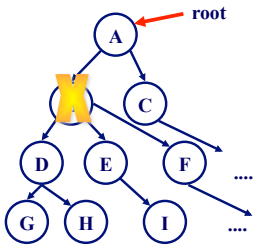
- T1 wants to insert in H



CMU SCS

Example B+tree

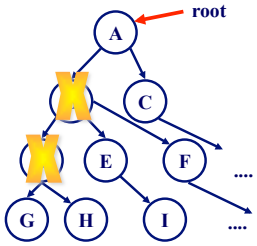
- T1 wants to insert in H
- (if 'B' is 'safe')



CMU SCS

Example B+tree

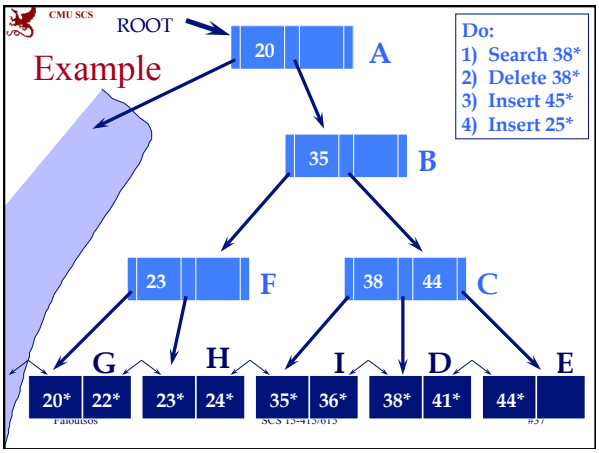
- T1 wants to insert in H
- continue 'crabbing'

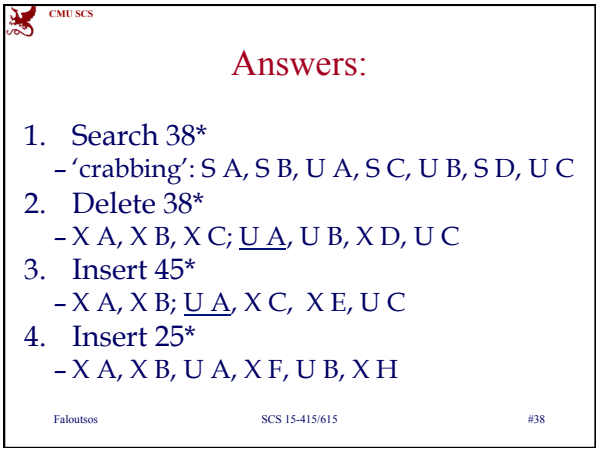


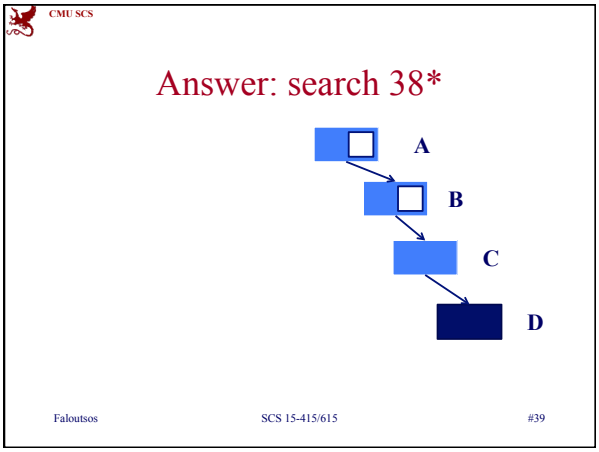
CMU SCS

A Simple Tree Locking Algorithm: "crabbing"

- 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.


CMU SCS
ROOT
Example
20
A
Do:
1) Search 38*
2) Delete 38*
3) Insert 45*
4) Insert 25*
35
B
23
F
38
44
C
G
H
I
D
E
20*
22*
23*
24*
35*
36*
38*
41*
44*

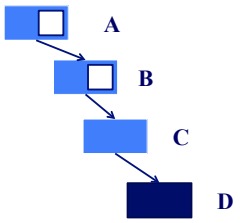

CMU SCS
Answers:
1. Search 38*
– 'crabbing': S A, S B, U A, S C, U B, S D, U C
2. Delete 38*
– X A, X B, X C; U A, U B, X D, U C
3. Insert 45*
– X A, X B; U A, X C, X E, U C
4. Insert 25*
– X A, X B, U A, X F, U B, X H
Faloutsos
SCS 15-415/615
#38


CMU SCS
Answer: search 38*
A
B
C
D
Faloutsos
SCS 15-415/615
#39

CMU SCS

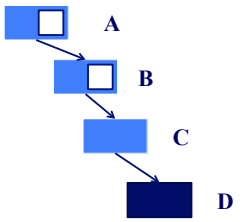
Answer: search 38*

S A
S B
U A
S C
U B
S D
U C
<read 38*>
U D



CMU SCS

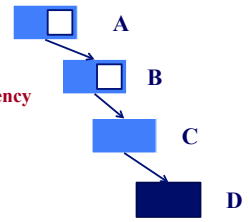
Answer: delete 38*



CMU SCS

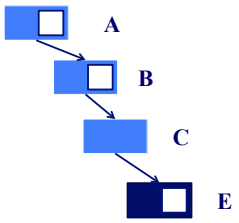
Answer: delete 38*

X A
X B
X C
U A ← **max concurrency**
U B
X D
U C
<delete 38*>
U D



CMU SCS

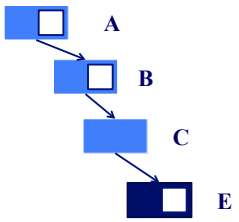
Answer: insert 45*



CMU SCS

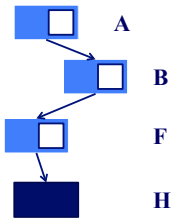
Answer: insert 45*

X A
X B
U A
X C
X E
U B
U C
<insert 45* >
U E



CMU SCS

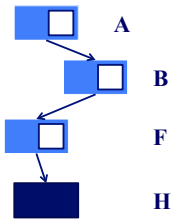
Answer: insert 25*



CMU SCS

Answer: insert 25*

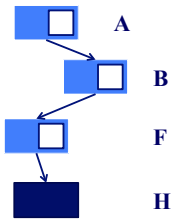
X A
X B
U A
X F
U B
X H
<insert 25*>
<split H>
<update F>
U F
U H



CMU SCS

Answer: insert 25*

X A
X B
U A
X F
U B
X H
<insert 25*>
<split H>
<update F>
U H
U F

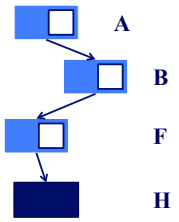


Q: Why not swap?

CMU SCS

Answer: insert 25*

X A
X B
U A
X F
U B
X H
<insert 25*>
<split H>
<update F>
U H
U F



Q: Why not swap?

A: swapping does not help concurrency!

CMU SCS

Answers:

1. Search 38*
   - 'crabbing': S A, S B, U A, S C, U B, S D, U C
2. Delete 38*
   - X A, X B, X C; U A, U B, X D, U C
3. Insert 45*
   - X A, X B; U A, X C, X E, U C
4. Insert 25*
   - X A, X B, U A, X F, U B, X H

CMU SCS

Answers:

1. Search 38*
   - 'crabbing': S A, S B, U A, S C, U B, S D, U C
2. Delete 38*
   - **X A** X B, X C; U A, U B, X D, U C
3. Insert 45*
   - **X A** X B; U A, X C, X E, U C
4. Insert 25*
   - **X A** X B, U A, X F, U B, X H

CAN WE DO BETTER?

CMU SCS

Can we do better?

- Yes [Bayer and Schkolnik]:
- Idea: hope that the leaf is 'safe', and use S-locks & crabbing to reach it, and verify
- (if false, do previous algo)

CMU SCS

Can we do better?

- Yes [Bayer and Schkolnik]:


Rudolf Bayer, Mario Schkolnick: *Concurrency of Operations on B-Trees*. Acta Inf. 9: 1-21 (1977)

CMU SCS

Can we do better?

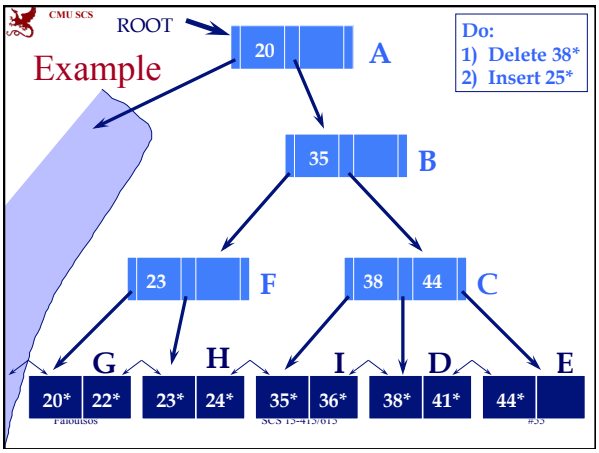
- Yes [Bayer and Schkolnik]:
- Main idea:
  - Gamble, that leaf is not over- (or under-) flowing
  - Thus, act as-if search, and only X-lock leaf, if bet is right
  - Otherwise, re-start, from top, with previous algo

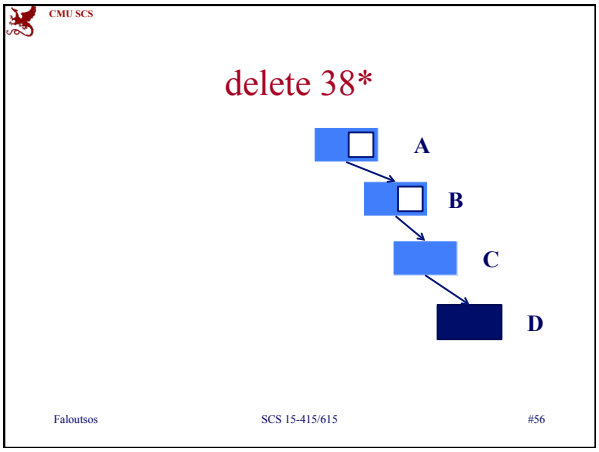
CMU SCS

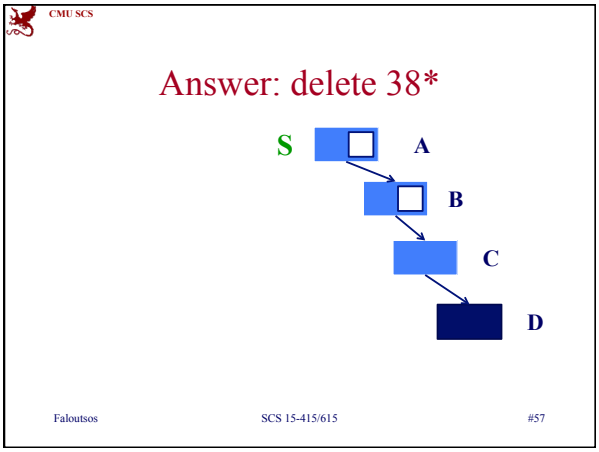
advanced

A Better Tree Locking Algorithm (From 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 alg.

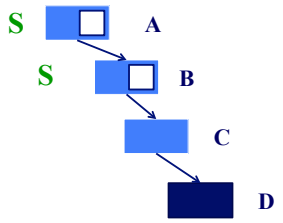

CMU SCS
ROOT
Example
20
A
Do:
1) Delete 38*
2) Insert 25*
35
B
23
F
38
44
C
G
H
I
D
E
20*
22*
23*
24*
35*
36*
38*
41*
44*


CMU SCS
delete 38*
A
B
C
D
Faloutsos
SCS 15-415/615
#56


CMU SCS
Answer: delete 38*
S
A
B
C
D
Faloutsos
SCS 15-415/615
#57

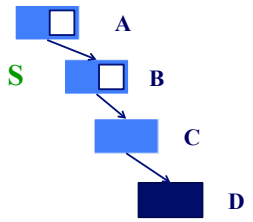
CMU SCS

Answer: delete 38*



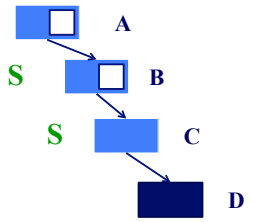
CMU SCS

Answer: delete 38*



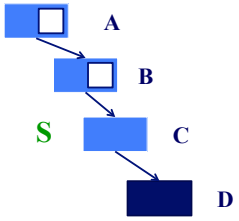
CMU SCS

Answer: delete 38*



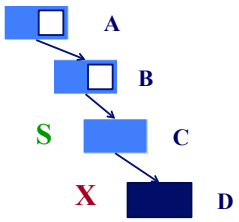
CMU SCS

Answer: delete 38*



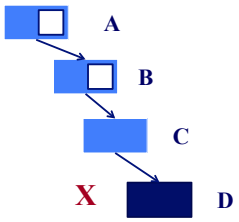
CMU SCS

Answer: delete 38*



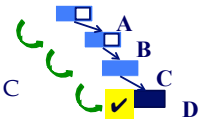
CMU SCS

Answer: delete 38*

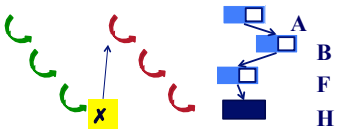


CMU SCS

Answers:



1. Delete 38*
   - S A, S B, U A, S C, U B, X D, U C
2. Insert 25*
   - S A, S B, U A, S F, U B, X H; U H;
   - **X A**, X B, U A, X F, U B, X H



CMU SCS

Notice:

- Textbook has a third variation, that uses lock-upgrades (and may lead to deadlocks)

CMU SCS

Outline

- Locking granularity
- Tree locking protocols
- Phantoms & predicate locking

Almost done with tree protocol

CMU SCS

A subtle point:

- Q1: Which order to release locks in multiple-granularity locking?
  - A1: bottom up
- Q2: Which order to release locks in tree-locking?
  - A2: as early as possible (to max concurrency)

CMU SCS

Outline

- Locking granularity
- Tree locking protocols
- Phantoms & predicate locking

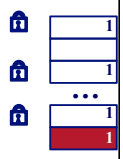
CMU SCS

Dynamic Databases – The "Phantom" Problem

- so far: only reads and updates – no insertions/ deletions
- with insertions/deletions, new problems:

CMU SCS

The phantom problem



time	T1	T2
71 ←	select max(age) ... where rating=1	
		insert ... age=96 rating=1
96 ←	select max(age) ... where rating=1	

CMU SCS

Why?

- because T1 locked only *existing* records – not ones under way!
- Solution?

CMU SCS

Solution

theoretical solution:

- 'predicate locking': e.g., lock all records (current or incoming) with rating=1
  - VERY EXPENSIVE

CMU SCS

Solution

practical solution:

- index locking: if an index (on 'rating') exists, lock the appropriate entries (rating=1 in our case)
- otherwise, lock whole table (and thus block insertions/deletions)

CMU SCS

Transaction Support in SQL-92

recommended

- **SERIALIZABLE** – No phantoms, all reads repeatable, no "dirty" (uncommited) reads.
- REPEATABLE READS – phantoms may happen.
- READ COMMITTED – phantoms and unrepeatable reads may happen
- READ UNCOMMITTED – all of them may happen.

CMU SCS

Transaction Support in SQL-92

- SERIALIZABLE : obtains all locks first; plus index locks, plus strict 2PL
- REPEATABLE READS – as above, but no index locks
- READ COMMITTED – as above, but S-locks are released immediately
- READ UNCOMMITTED – as above, but allowing 'dirty reads' (no S-locks)

CMU SCS

Transaction Support in SQL-92

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE READ ONLY

Defaults:

SERIALIZABLE ← **isolation level**

READ WRITE ← **access mode**

CMU SCS

Summary

- Multiple granularity locking: leads to few locks, at appropriate levels
- Tree-structured indexes:
  - 'crabbing' and 'safe nodes'
- (notice:
  - Multiple gran. locking: releases locks bottom-up
  - Tree-locking: top-down (to max. concurrency)

CMU SCS

Summary

- "phantom problem", if insertions/deletions
  - (Predicate locking prevents phantoms)
  - Index locking, or table locking