Transaction Management Exercises KEY

I/O and CPU activities can be and are overlapped to minimize (disk and processor) idle time and to maximize throughput (units of “work” per time unit). This motivates concurrent, interleaved execution of transactions.

Some of these exercises will be more or less review of the week’s reading and videos, while others go beyond the previous work (e.g., locking, which is only touched on by the text).

Study Group Participants:

1. Consider the following two transactions, T1 and T2:

T1: Read(A), Op_{11}(A), Write(A), Read(B), Op_{12}(B), Write(B), Commit

T2: Read(A), Op_{21}(A), Write(A), Read(B), Op_{22}(B), Write(B), Commit

Three interleaved *schedules* are (just showing disk Reads and Writes):

S1

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	Commit
Commit	

S2

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

S3

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

The questions on the next page could apply whether the arguments (A and B) to the operators, Read, Write, Op_{ij} were tables or other database entity, but you might ground the question by assuming that A and B are tuples (or records), say in a table of banking accounts, where T1 implements a transfer of funds from account A to account B. So Op_{11}(A) decrements A by $100, and Op_{12} increments B by $100), and T2 updates each account by 10% interest – that is both Op21 and Op22 increment their respective accounts by 10%.

1. Consider the following two transactions, T1 and T2:

T1: Read(A), $Op_{11}(A)$, Write(A), Read(B), $Op_{12}(B)$, Write(B), Commit

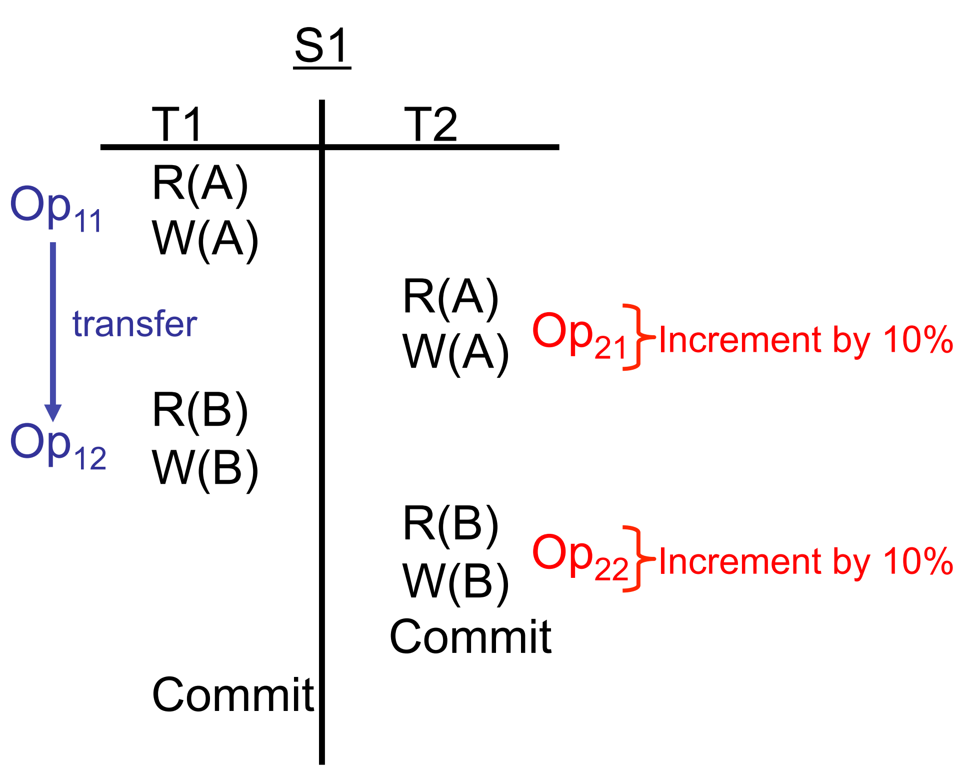
T2: Read(A), $Op_{21}(A)$, Write(A), Read(B), $Op_{22}(B)$, Write(B), Commit

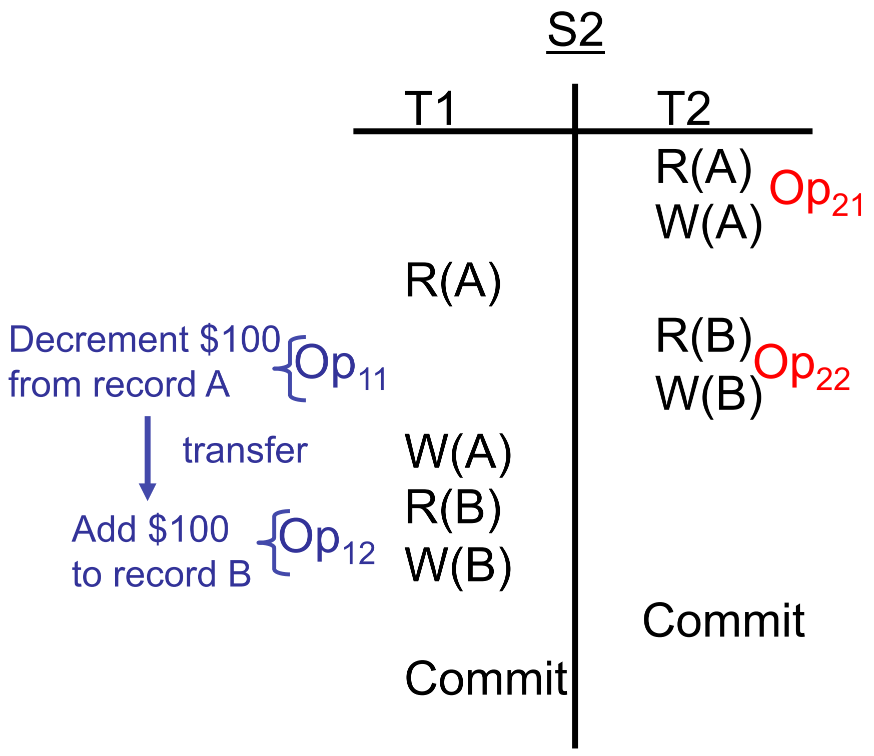
Three interleaved *schedules* are (just showing disk Reads and Writes):

a) Using the banking example from the previous page, understand each schedule in terms of that example, replacing each Op_{ij} with the respective increment, decrement, and interest increment operators.

b) Which of the schedules (S1, S2, S3) is guaranteed equivalent to a serial execution? In these cases, give the equivalent execution order:

c) Which of these schedules includes an example of a dirty read?



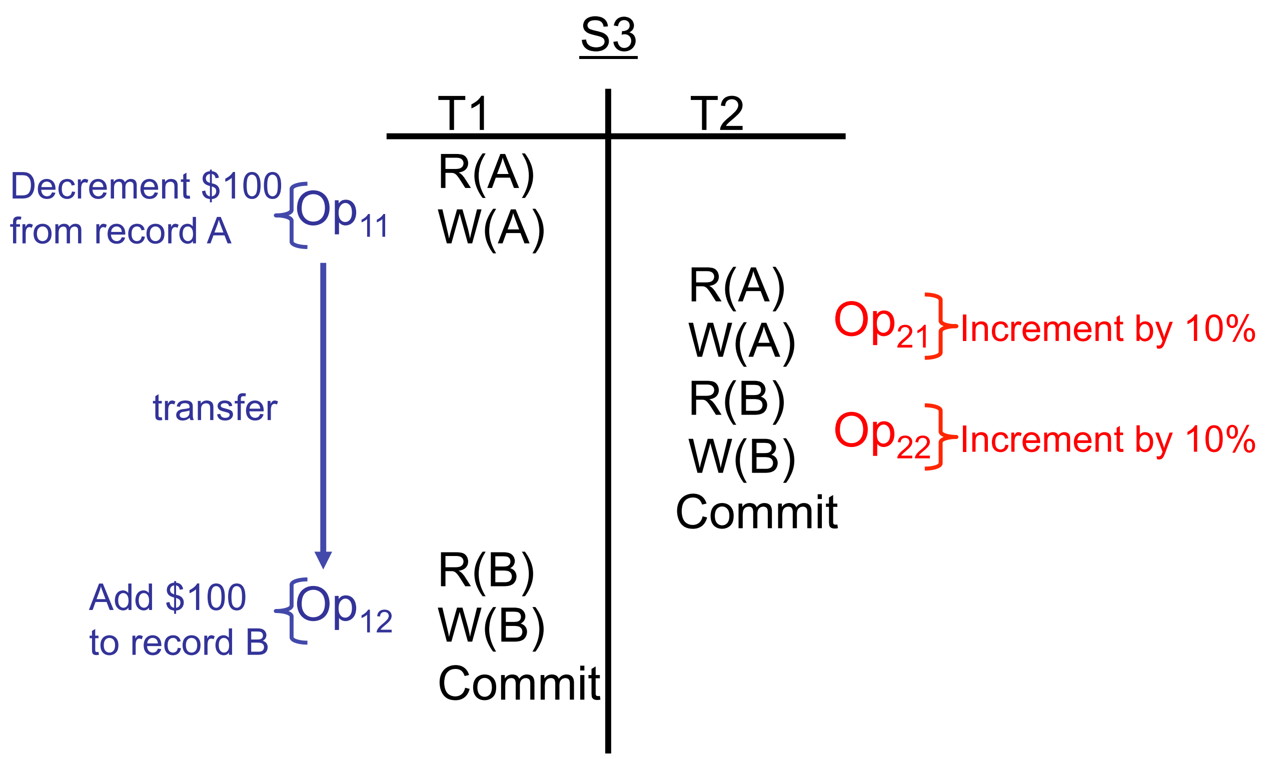


S1 is serializable:
T1 ➔ T2

S2 is serializable:
T2 ➔ T1

Different serializations, T1➔T2 and T2➔T1, need not lead to the same DB instances.

Example above: incrementing accounts by 10% after transfer (S1) versus before transfer (S2)



S3 is *not* serializable: it *may* yield different results than either T1➔T2 or T2➔ T1

10% increments are made on both tables at lowest value. In general, this can be a problem (sometimes) with dirty reads (when one transaction reads data that has been changed by another transaction prior to that other transaction commiting).

S1

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	Commit
Commit	

S2

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

S3

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

S1 is *serializable*: it yields the same result as T1 run to completion, followed by T2 run to completion, or T1 ➔ T2

S2 is serializable: T2 ➔ T1

S3 is *not* serializable: it *may* yield different results than either T1➔T2 or T2➔ T1

Each of these schedules has examples of a *dirty read,* which can sometimes lead to anomolies.

Now for a new topic: *strict two-phase locking* (2PL)

1. If a Transaction wants to ONLY *READ* an “object” (e.g., tuple, table, index node), it first obtains a *shared* lock on the “object” (shared, because others can do so too)

2. If a Transaction wants to *MODIFY/WRITE* an “object” it first obtains an *exclusive* lock on the “object”

3. *All locks held by a transaction are released when the transaction is complete (upon Commit)*

A shared lock on an object can be obtained in the absence of an exclusive lock on the object *by another transaction*.

An exclusive lock can be obtained in the absence of any lock by another transaction

Basically, locking is concerned with insuring atomic and isolation properties of individual transactions, while exploiting parallelism/interleaving.

Lets return to the earlier abstract two-transaction example, but now with locks added to new schedules, S4-S6 (don’t show Op_{ij} explicitly)

S4

T1	T2
X(A)	
R(A)	
W(A)	
X(B)	
R(B)	
W(B)	
Commit	
	X(A)
	R(A)
	W(A)
	X(B)
	R(B)
	W(B)
	Commit

S5

T1	T2
S(A)	
R(A)	
	S(A)
	R(A)
X(A)	
W(A)	
X(B)	
R(B)	
W(B)	
Commit	
	X(A)
	W(A)
	X(B)
	R(B)
	W(B)
	Commit

S6

T1	T2
X(A)	
R(A)	
W(A)	
	X(A)
	R(A)
	W(A)
X(B)	
R(B)	
W(B)	
Commit	
	X(B)
	R(B)
	W(B)
	Commit

X: exclusive lock S: shared lock

Which of the schedules above are viable under 2PL? Inversely, which of the schedules above are ILLEGAL under 2PL, because they violate locking protocols of previous slide? In which schedules does deadlock (i.e., an inability to advance) occur

S4

T1	T2
X(A)	
R(A)	
W(A)	
X(B)	
R(B)	
W(B)	
Commit	
	X(A)
	R(A)
	W(A)
	X(B)
	R(B)
	W(B)
	Commit

OK

S5

T1	T2
S(A)	
R(A)	
	S(A)
	R(A)
~~X(A)~~	
W(A)	
X(B)	
R(B)	
W(B)	
Commit	
	X(A)
	W(A)
	X(B)
	R(B)
	W(B)
	Commit

deadlock

Illegal under Strict 2PL

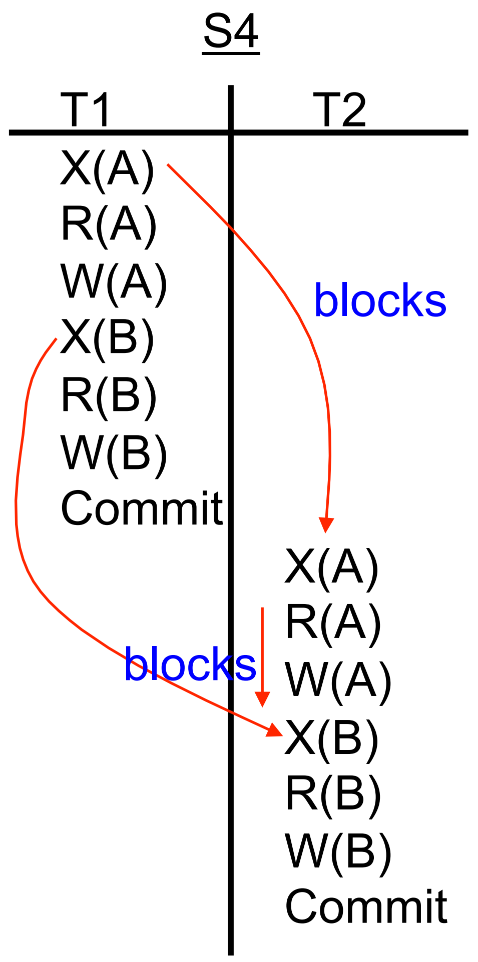
NO

S6

T1	T2
X(A)	
R(A)	
W(A)	
	~~X(A)~~
	R(A)
	W(A)
X(B)	
R(B)	
W(B)	
Commit	
	X(B)
	R(B)
	W(B)
	Commit

NO

Follow-up: Can we do any interleaving of T1 and T2 under strict 2PL at all?



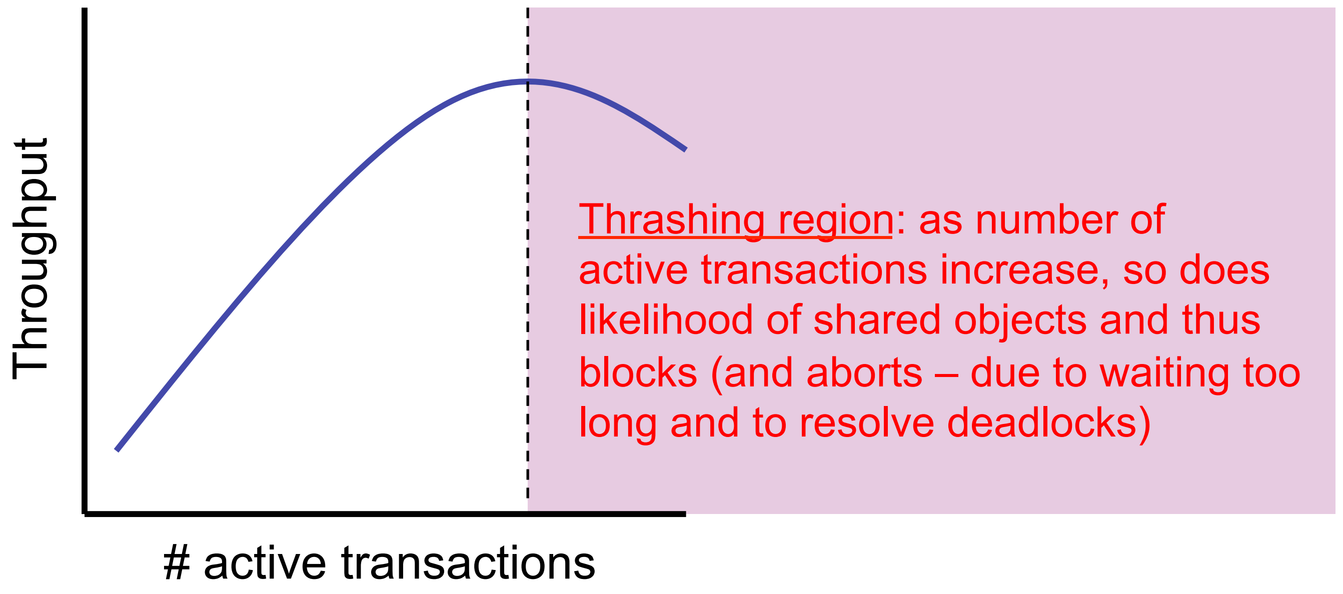
Can you do any interleaving of T1 and T2 under strict 2PL at all?

In cases where transactions involve the same objects, Strict 2PL can radically limit opportunities for parallelism/interleaving

…. But Strict 2PL makes interleaving safe, and the good news is that ….

in practice, there are many transactions that do not involve the same objects and that can be interleaved to improve throughput

and even transactions that share objects (through reads) can be interleaved with strict 2PL (and shared locks)



What “objects” can be locked?

Entire tables

Individual records within a table

A set of records that satisfy a condition (e.g., TransNumber = abc)

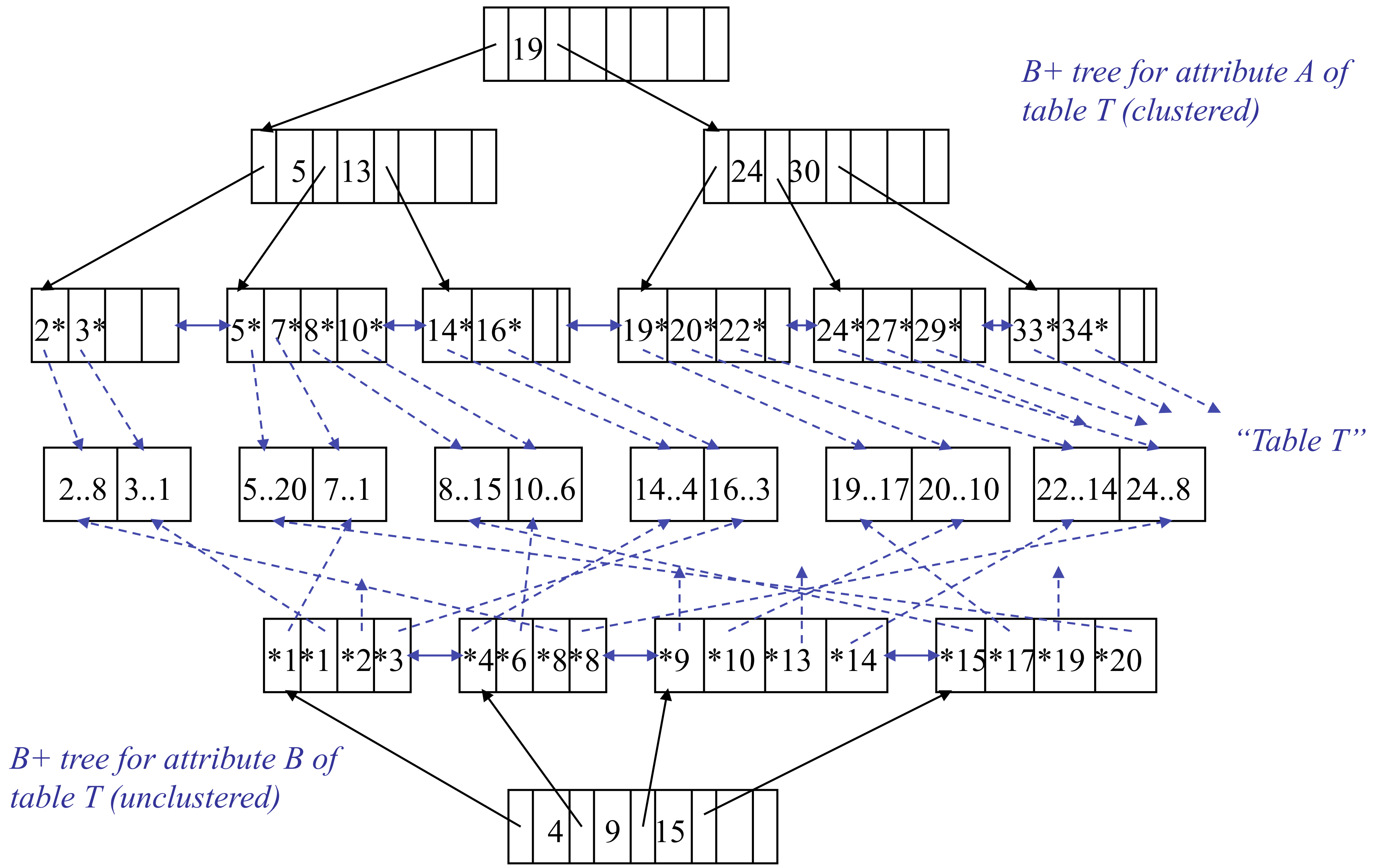
An entire indexing structure on an attribute for a table

Individual nodes (index pages) within the indexing structure (next exercise)

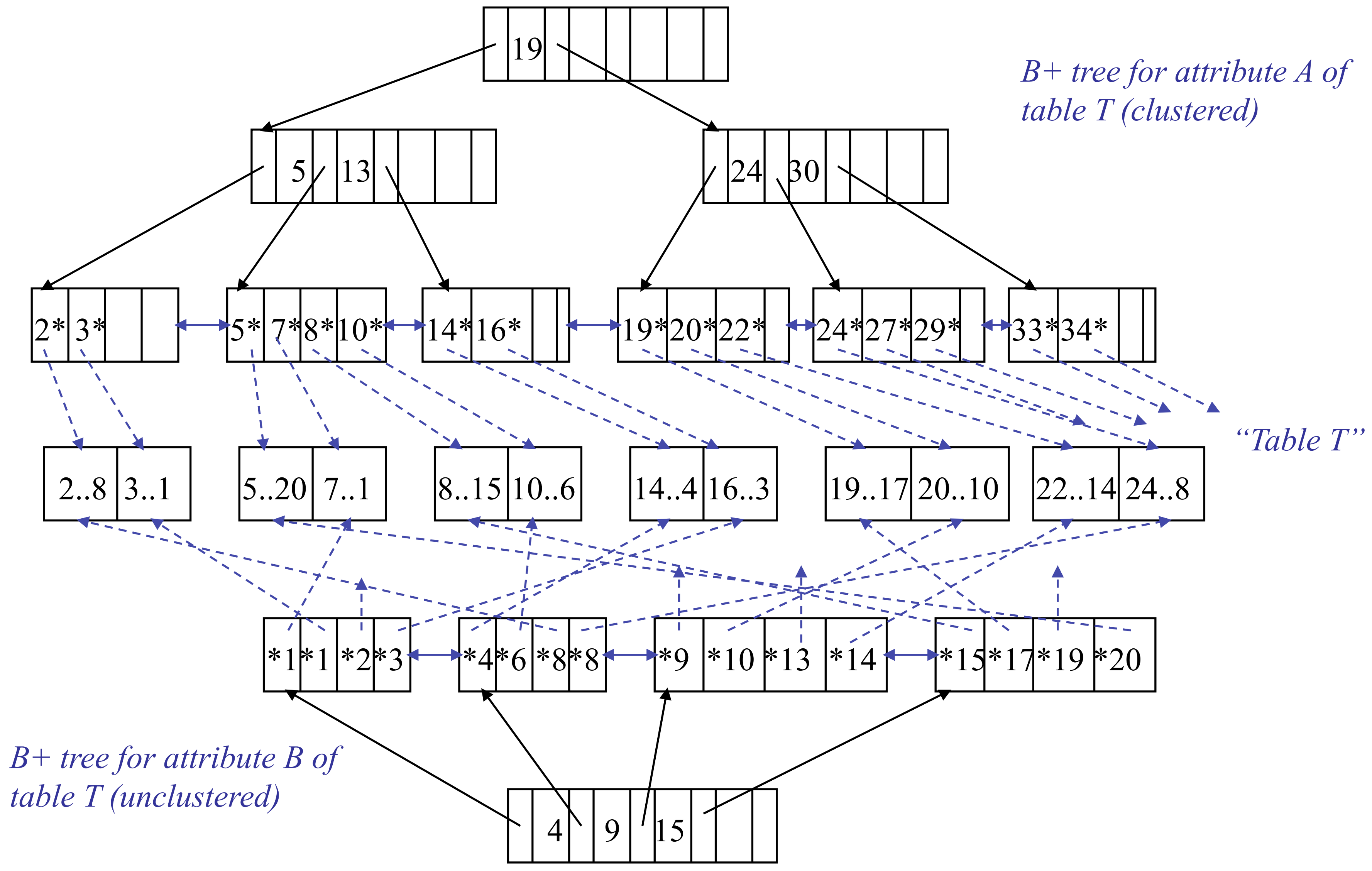
Individual data pages (next exercise)

In general, we want exclusive locks on the smallest “objects” possible?

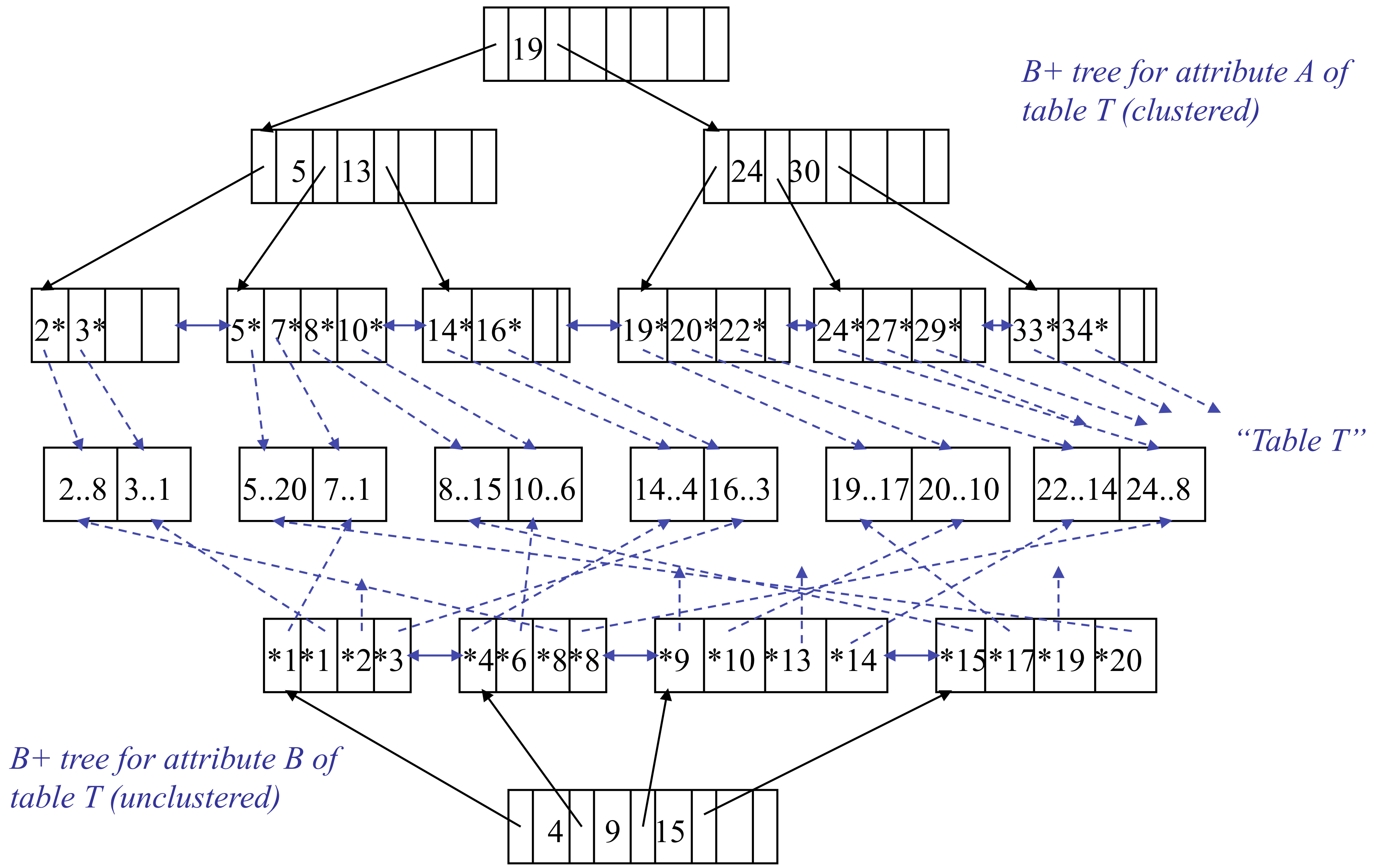
Can individual attribute fields of an individual record be locked?
Check it out for yourself – what do you think?



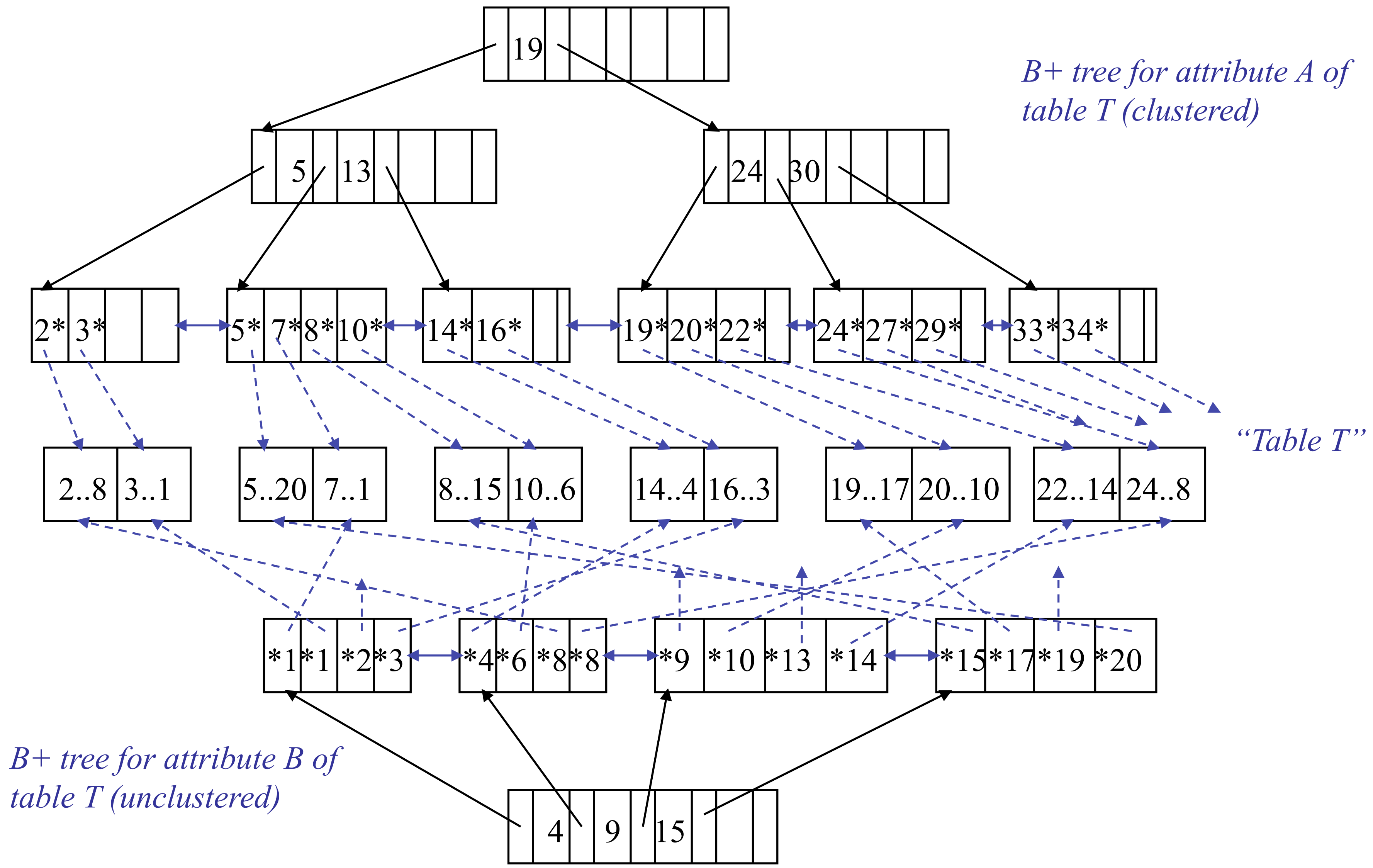
Given these indexes, identify the shared and exclusive locks that each of the SQL commands here and on subsequent pages would require.



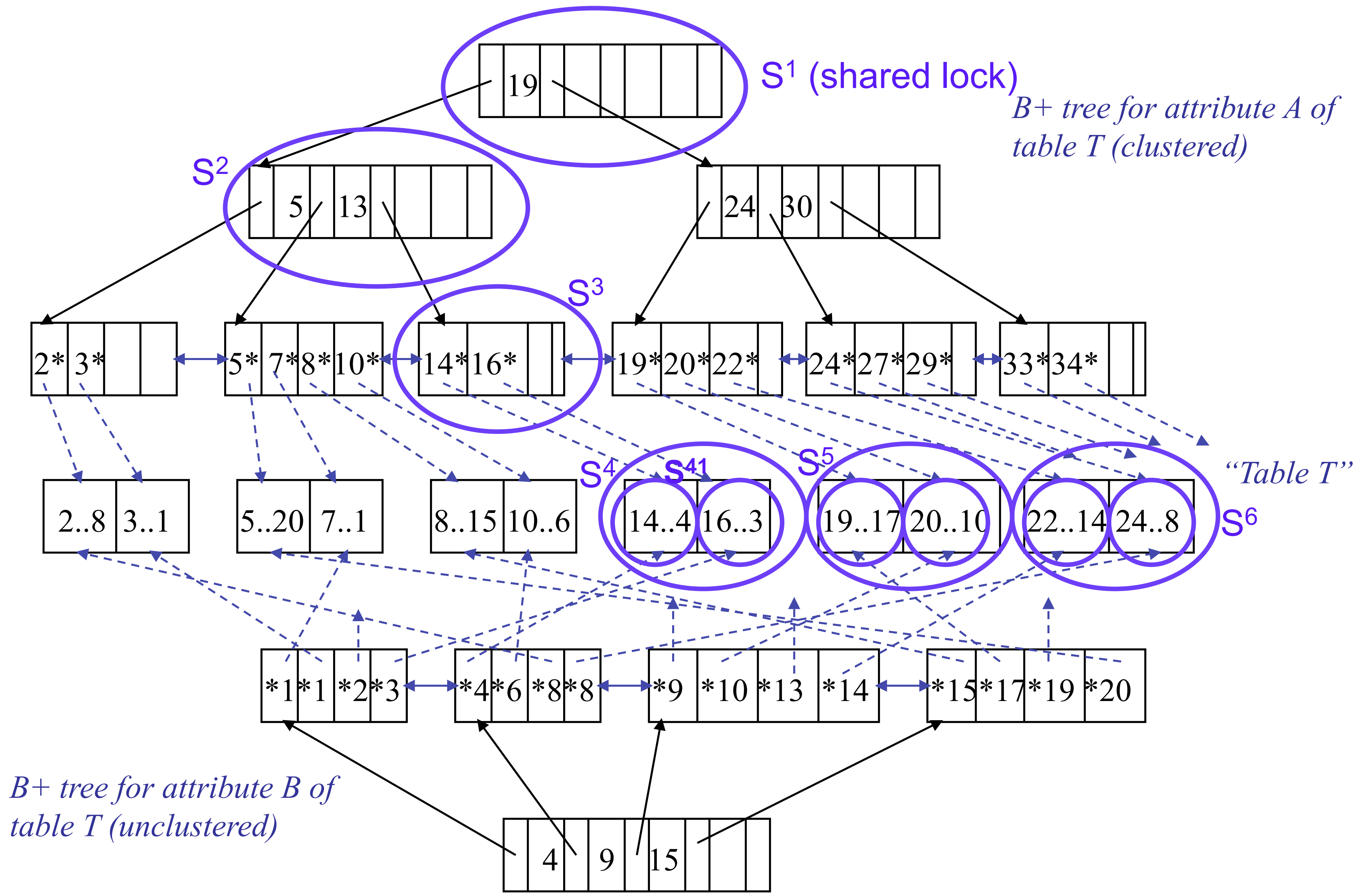
Given these indexes, identify the shared and exclusive locks that this SQL command would require: **SELECT T.C FROM T WHERE T.A > 14 AND T.B <= 10** (hint: the query evaluator would probably only use the attribute A index for this)



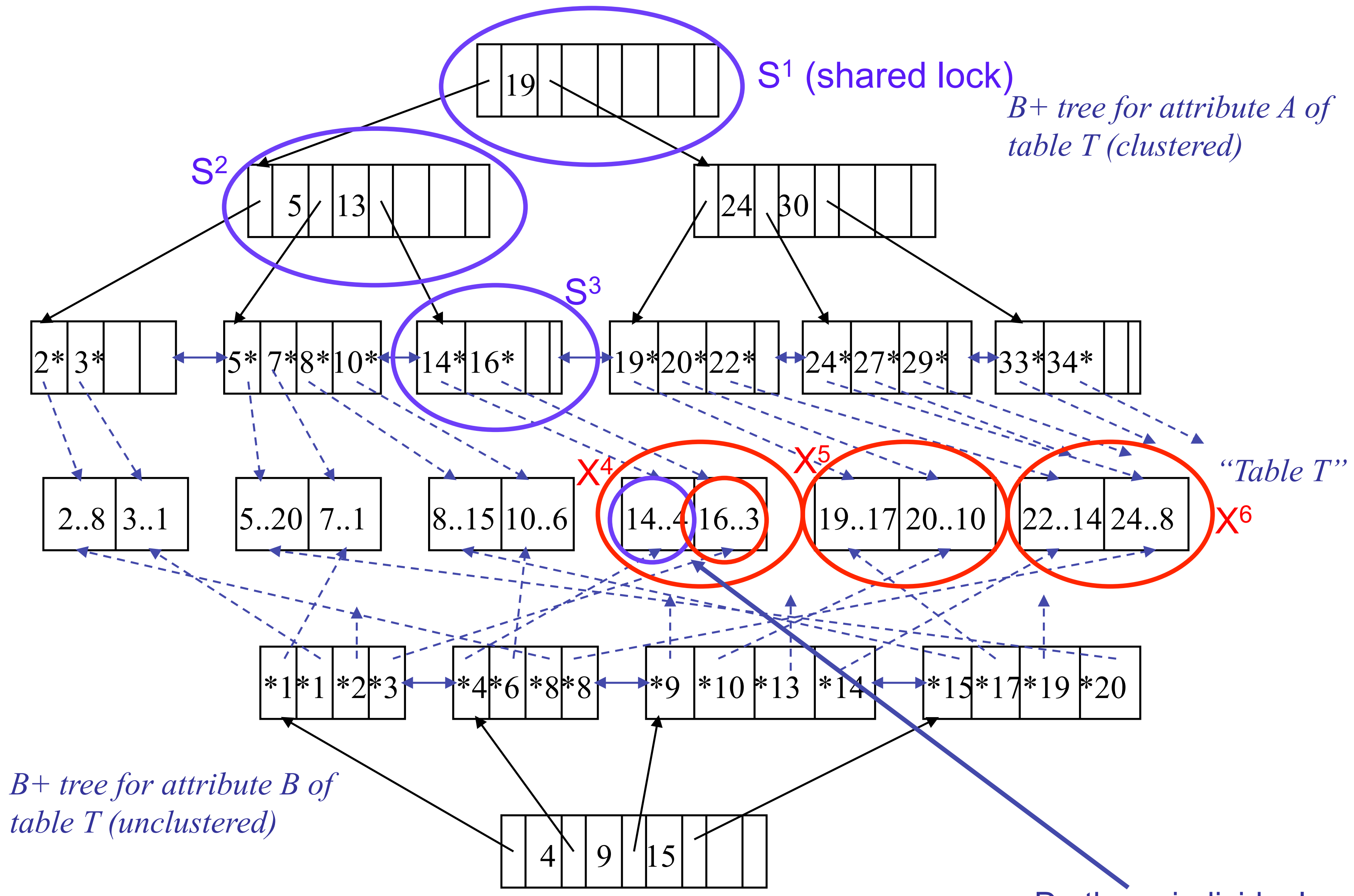
Given these indexes, identify the shared and exclusive locks that this SQL command would require: **UPDATE T SET T.C = T.C+1 WHERE T.A > 14 AND T.B <= 10** (hint: the query evaluator would probably only use the attribute A index for this)



Given these indexes, identify the shared and exclusive locks that this SQL command would require:
INSERT INTO T (A, B, C) VALUES (18, 9, 12) (hint: both indexes involved)

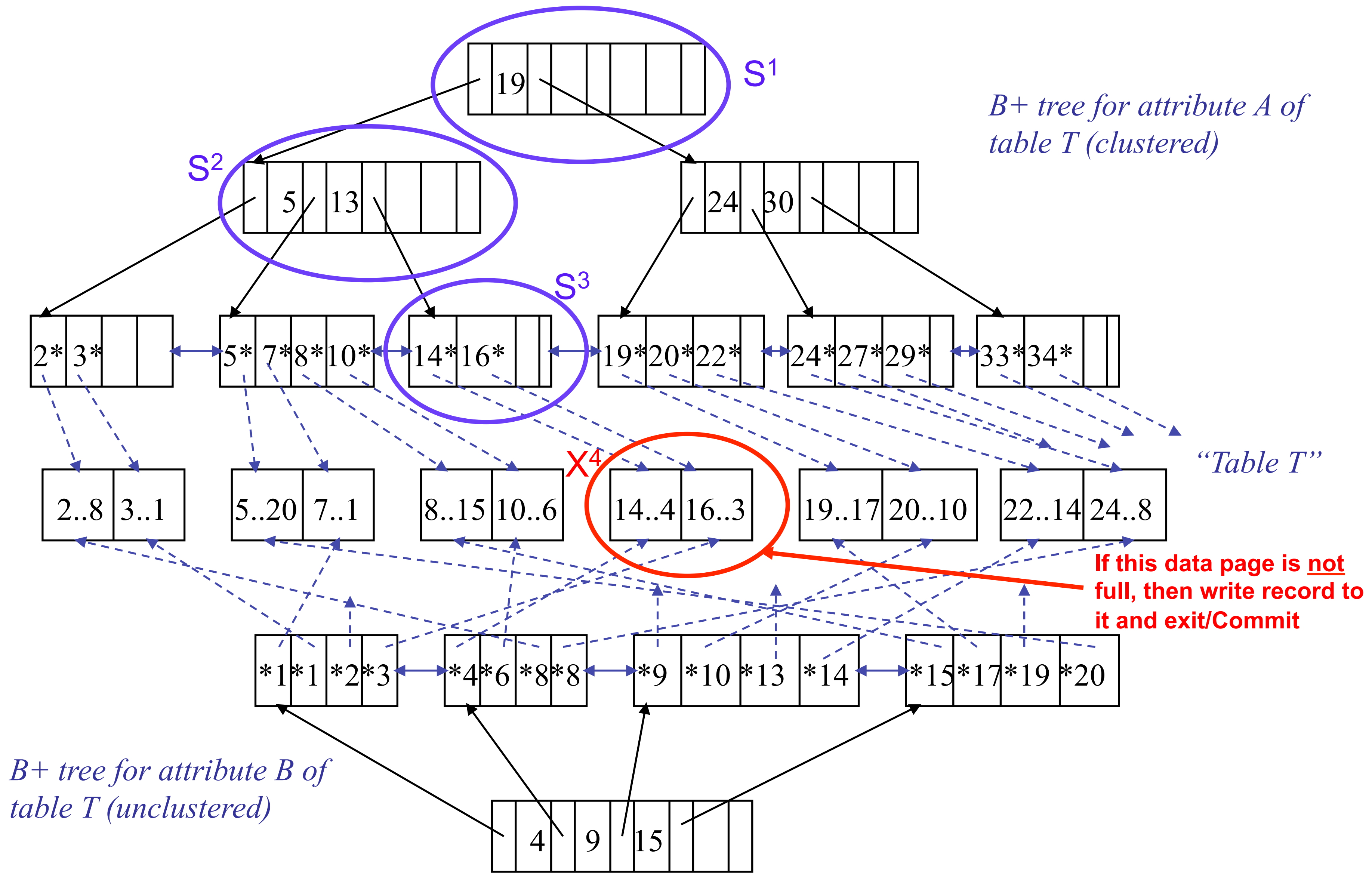


SELECT T.C FROM T WHERE **T.A > 14** AND T.B <= 10



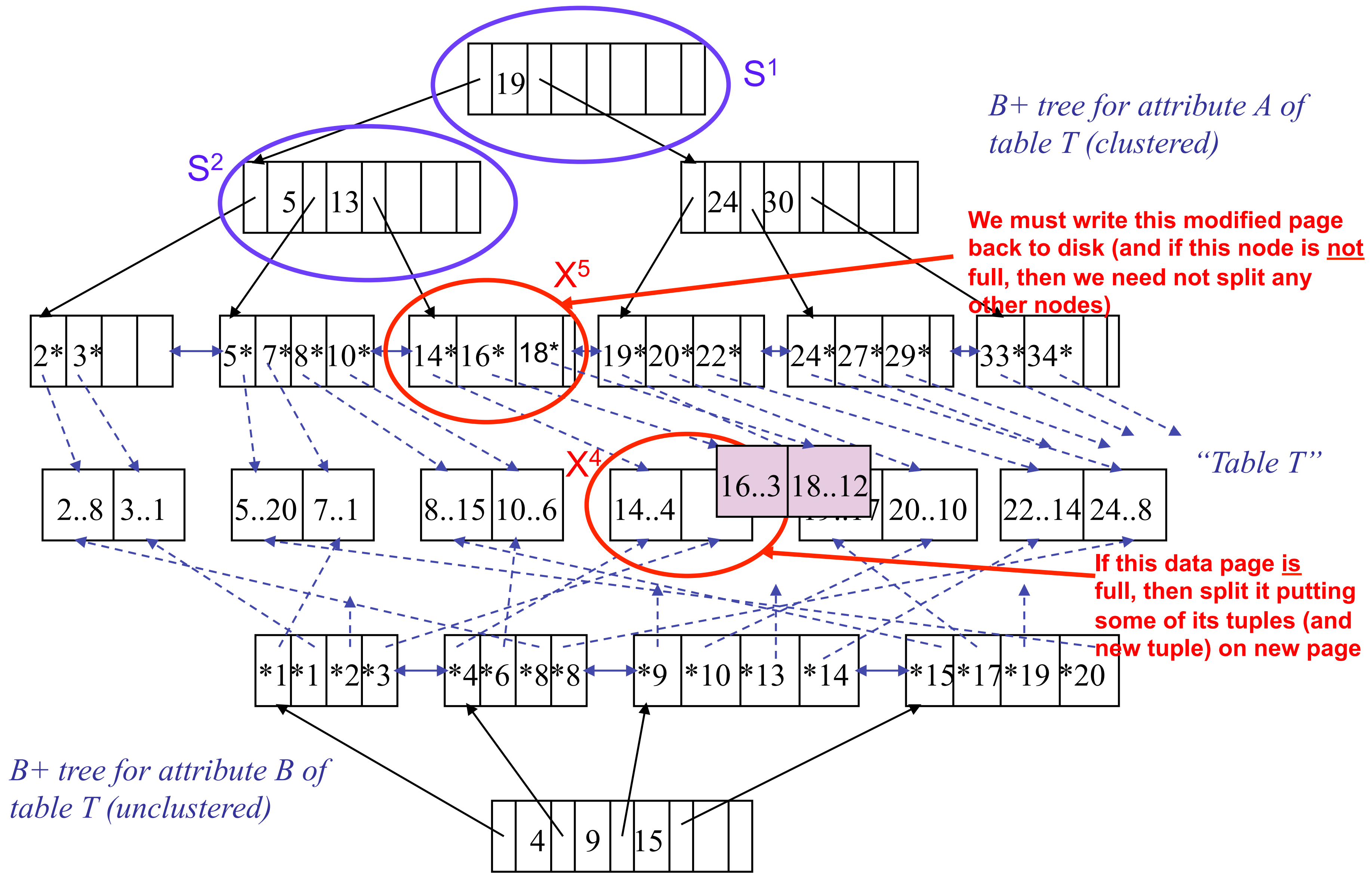
Do these individual record locks make sense given the page locks?

UPDATE T SET T.C = T.C+1 WHERE **T.A > 14** AND T.B <= 10



INSERT INTO T (A, B, C) VALUES (**18**, **12**, 9)

Continued on next page



B+ tree for attribute A of table T (clustered)

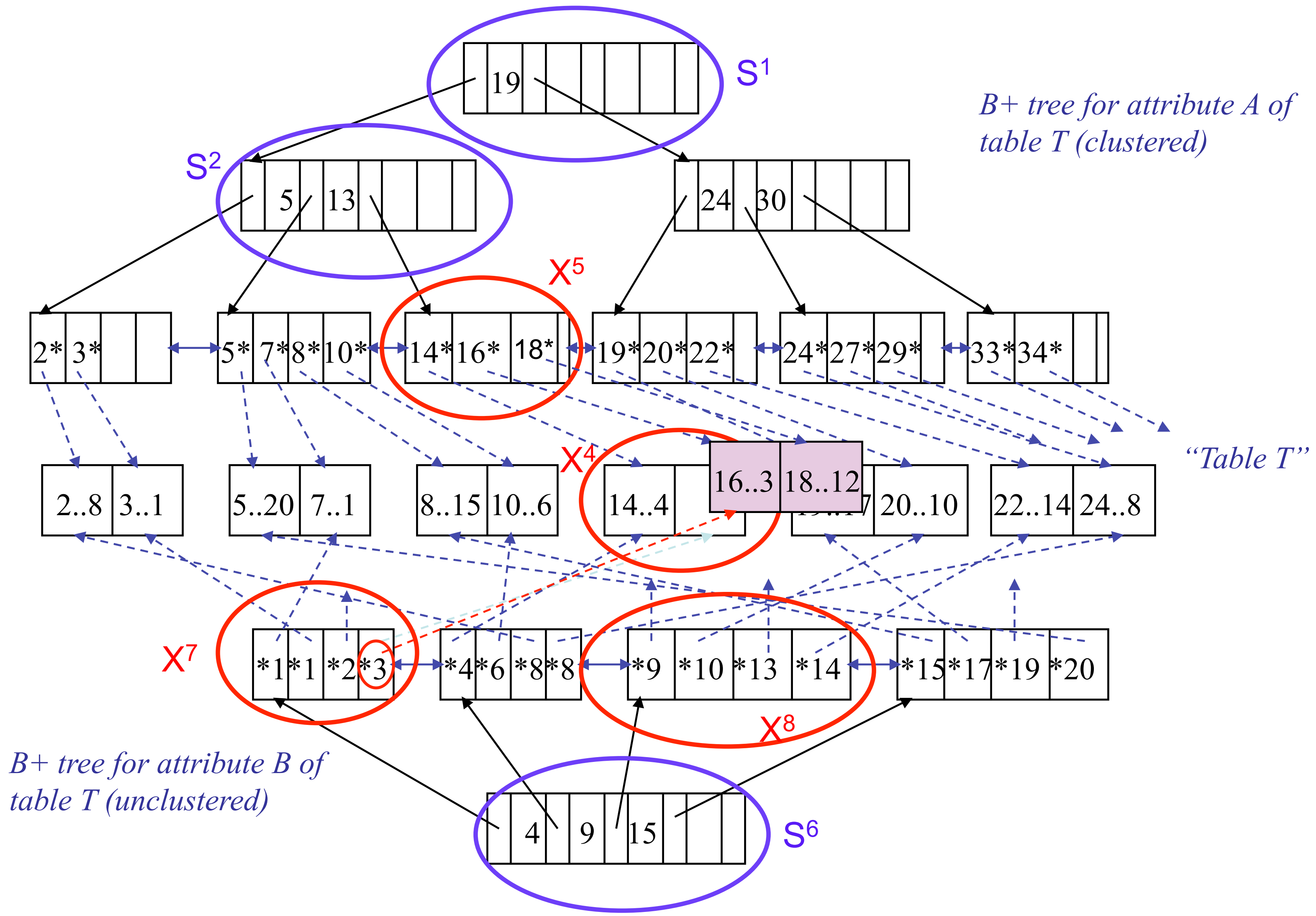
We must write this modified page back to disk (and if this node is not full, then we need not split any other nodes)

If this data page is full, then split it putting some of its tuples (and new tuple) on new page

B+ tree for attribute B of table T (unclustered)

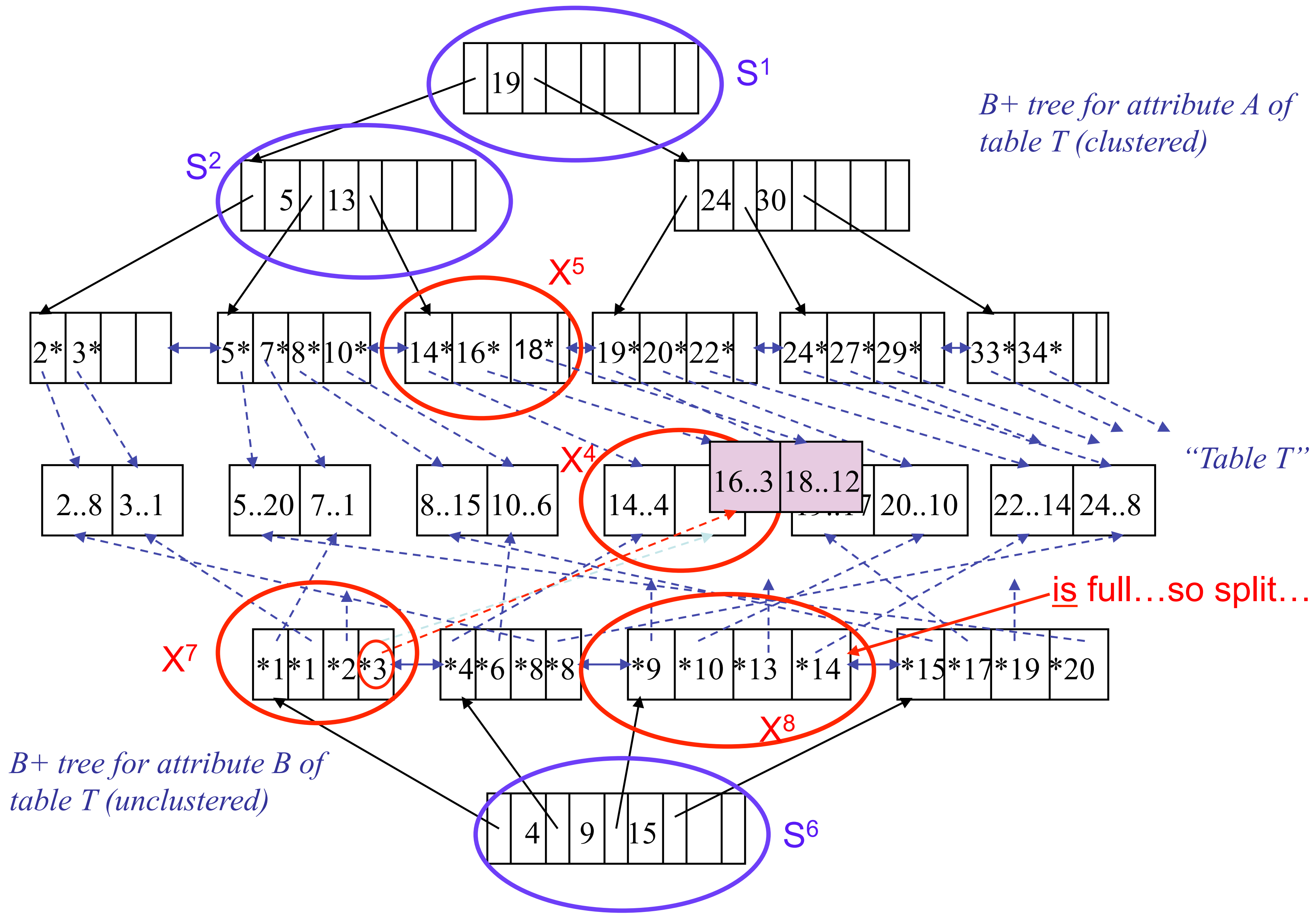
INSERT INTO T (A, B, C) VALUES (**18, 12**, 9)

Continued on next page



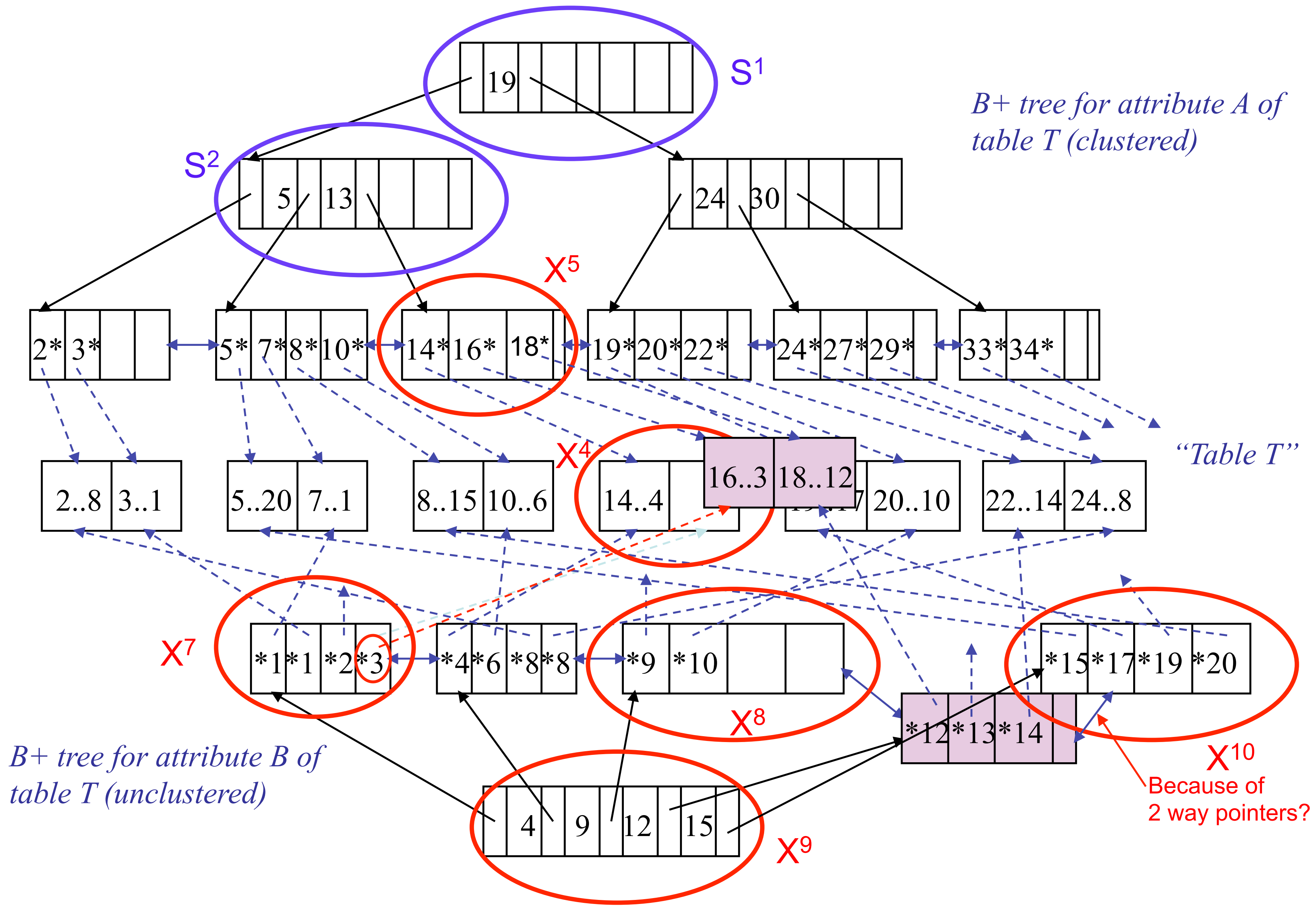
INSERT INTO T (A, B, C) VALUES (**18**, **12**, 9)

Continued on next page



INSERT INTO T (A, B, C) VALUES (**18**, **12**, 9)

Continued on next page



INSERT INTO T (A, B, C) VALUES (**18, 12**, 9)