Concurrency Control 3

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

Overview

- Deadlock Handling
- Multiple Granularity
- Assignments

Deadlock Handling

- System is **deadlocked** if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.
- Recall the following example:

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Deadlock Handling

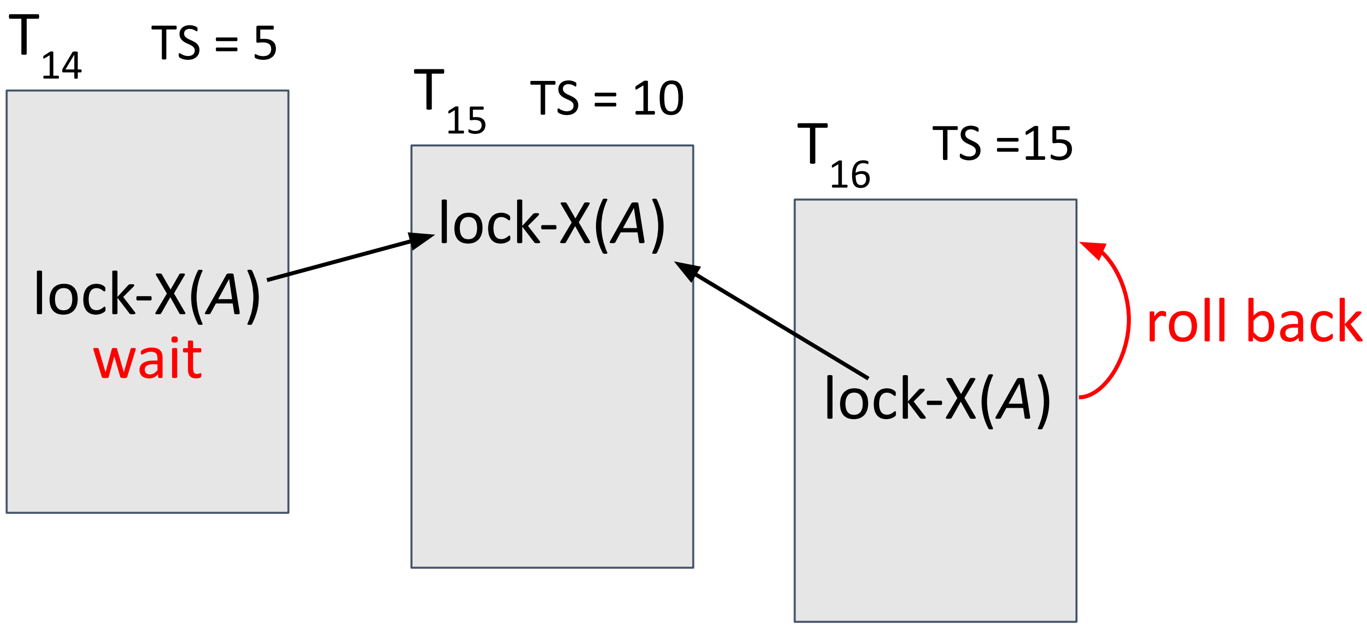
- There are two principal methods for deadlock handling
    - ***Deadlock prevention***: ensuring that the system will *never* enter into a deadlock state
    - ***Deadlock detection*** and ***deadlock recovery***: allowing the system to enter into a deadlock state,but detect the deadlock when it occurs and recovery is made to resolve the deadlock situation
- Both methods may result in transaction rollback
- Prevention is commonly used if the deadlock likely to occur frequently; otherwise, detection and recovery are more efficient

Deadlock Prevention

- ***Deadlock prevention*** strategies:
  - Pre-declaration: requiring that each transaction locks all its data items before it begins execution
  - Graph-based protocol: Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order
    - Two-phase locking can be used with slight modification in a similar way

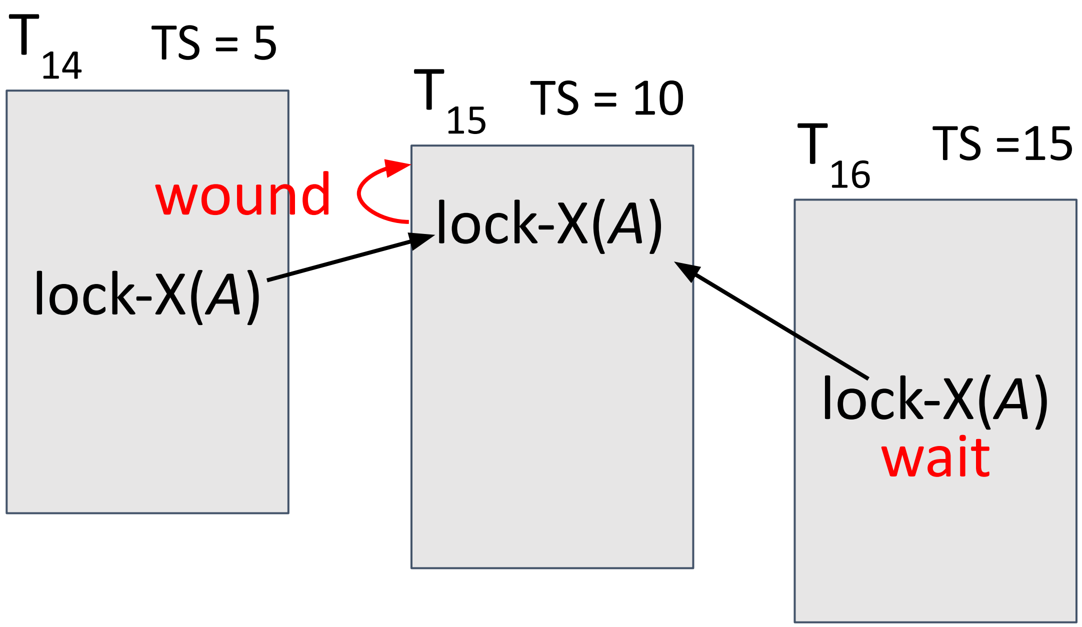
Deadlock Prevention (Cont.)

- ***Deadlock prevention*** strategies (Cont.):
  - **wait-die** scheme — non-preemptive
    - Older transaction may wait for younger one to release data item.
    - Younger transactions never wait for older ones; they are rolled back instead.



Deadlock Prevention (Cont.)

- ***Deadlock prevention*** strategies (Cont.):
  - **wound-wait** scheme — preemptive
    - Older transaction *wounds* (forcefully roll back) younger transaction instead of waiting for it.
    - Younger transactions may wait for older ones.

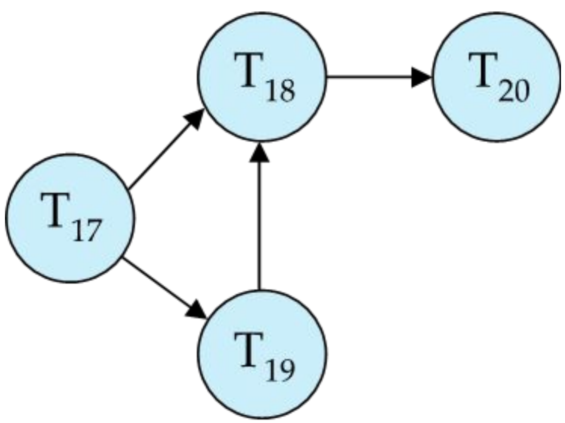


Deadlock Prevention (Cont.)

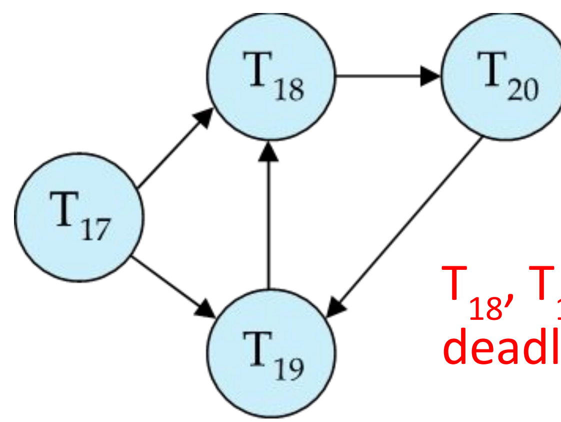
- **Timeout-Based Schemes**:
  - A transaction waits for a lock only for a specified amount of time. After that, the wait times out and the transaction is rolled back.
  - Ensures that deadlocks get resolved by timeout if they occur
  - Simple to implement
  - But may roll back transaction unnecessarily in absence of deadlock
    - Difficult to determine good value of the timeout interval.
  - Starvation is also possible

Deadlock Detection

- **Wait-for graph**
  - *Vertices:* transactions
  - *Edge from* $T_i \rightarrow T_j$. : if T_i is waiting for a lock held in conflicting mode by T_j
- The system is in a deadlock state if and only if the wait-for graph has a cycle.
  - Transactions involved in the cycle are deadlocked



Wait-for graph without a cycle



T_{18}, T_{19}, T_{20} are deadlocked

Wait-for graph with a cycle

Deadlock Recovery

- When deadlock is detected :
  - Some transaction will have to be rolled back (**victim**) to break the deadlock cycle.
    - Select that transaction as victim that will incur minimum cost
  - Rollback -- determine how far to roll back transaction
    - **Total rollback**: Abort the transaction and then restart it.
    - **Partial rollback**: Roll back victim transaction only as far as necessary to release locks that another transaction in cycle is waiting for
- Starvation can happen if the victim transaction happens to be always the one that incurs minimum cost
  - One solution: when we chose a victim, consider not only the cost but also the number of rollbacks that has been done

Overview

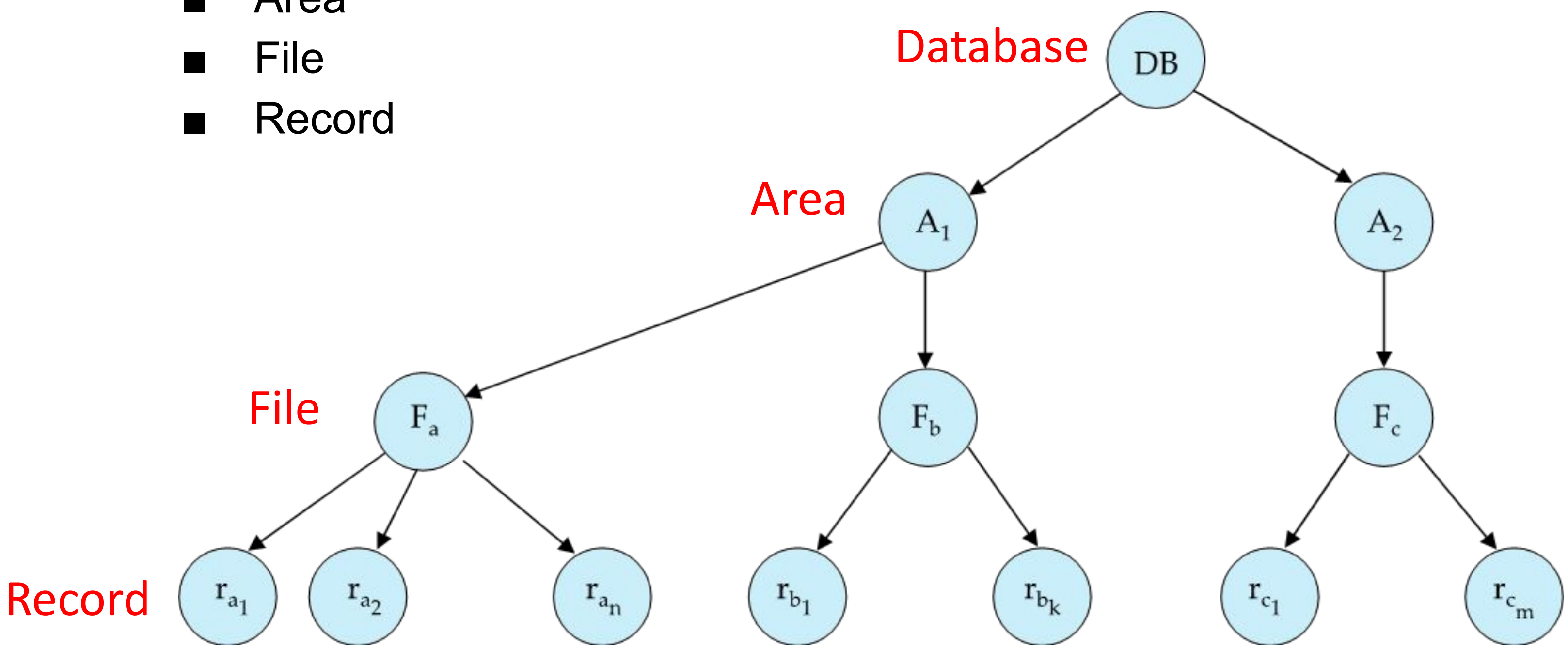
- Deadlock Handling
- Multiple Granularity
- Assignments

Multiple Granularity

- If a transaction wants to access most of tuples in the relation, locking each tuple is time-consuming and takes up too much memory
  - We may want to allow locking the entire relation
    - Reducing the degree of concurrency
- Granularity of locking:
    - **Fine granularity**: high concurrency, high locking overhead
    - **Coarse granularity**: low locking overhead, low concurrency
- Multiple granularity of locking:
  - Various sizes for data items
  - Define a hierarchy of data granularities
    - Small granularities are nested within larger ones
    - Hierarchy of data granularities can be represented graphically as a tree (don't confuse with tree-locking protocol)

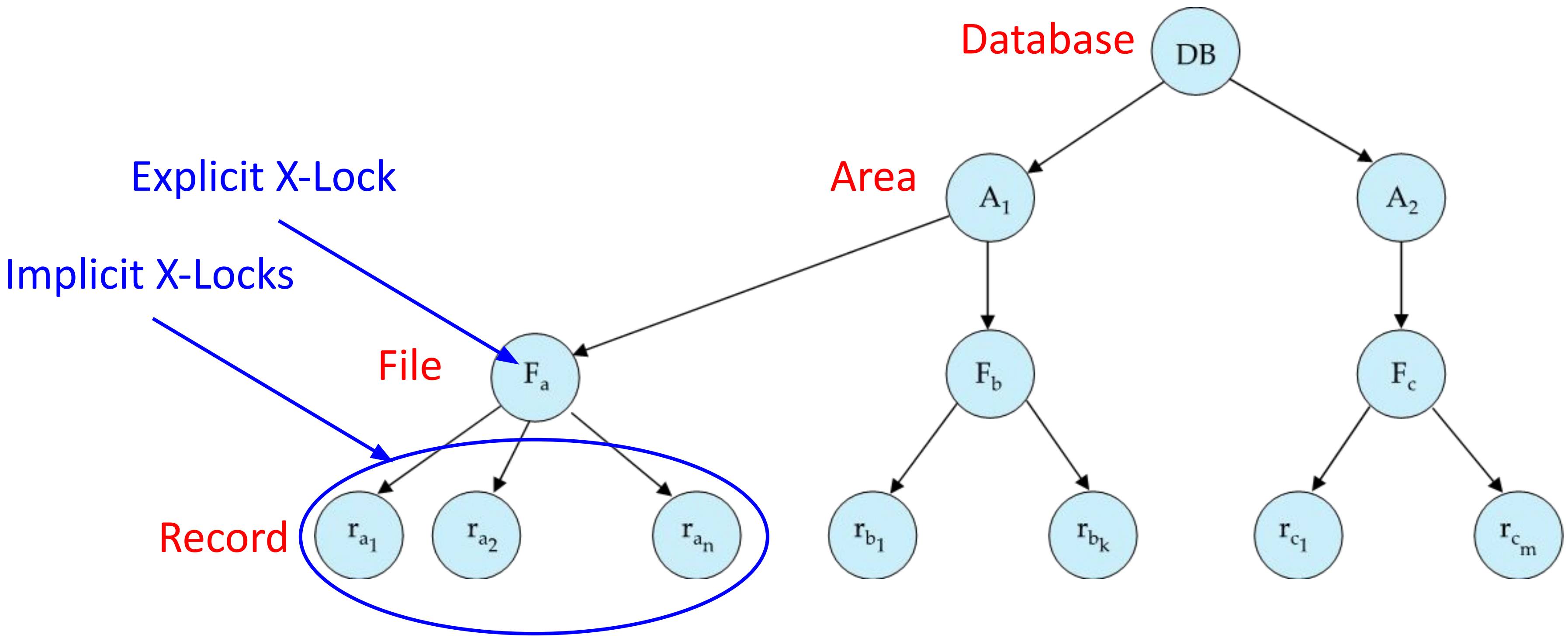
Example of Granularity Hierarchy

- Example:
  - Below we have a tree for the following granularity hierarchy of data items where each non-leaf node is a data granularity represented by its children
    - Database
    - Area
    - File
    - Record



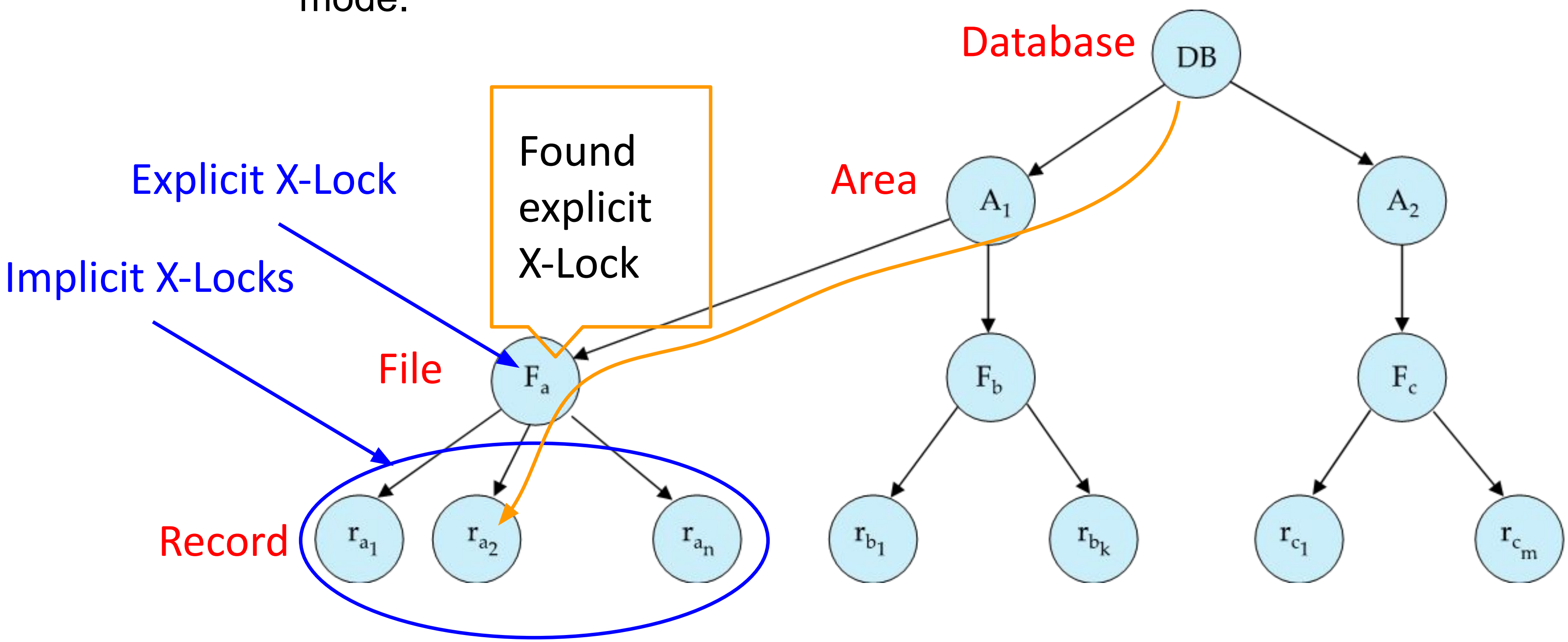
Example of Granularity Hierarchy (Cont.)

- When a transaction locks a node in the tree *explicitly*, it *implicitly* locks all the node's descendants in the same mode.



Example of Granularity Hierarchy (Cont.)

- Example:
  - If a transaction wishes to lock a record, it should check if the record is locked or not.
  - To do so, the transaction traverse the tree from the root to the target record and check if any node in that path is locked in an incompatible mode.



Intention Lock Modes

- In addition to S and X lock modes, there are three additional lock modes with multiple granularity:
    - ***intention-shared*** (IS): indicates explicit locking at a lower level of the tree but only with shared locks.
    - ***intention-exclusive*** (IX): indicates explicit locking at a lower level with exclusive or shared locks
    - ***shared and intention-exclusive*** (SIX): the subtree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.
- Intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes.

Compatibility Matrix with Intention Lock Modes

- The compatibility matrix for all lock modes is:

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

Overview

- Deadlock Handling
- Multiple Granularity
- Assignments

Assignments

- Reading: Ch18.2, 18.3
- Practice Excercises: 18.9

Solutions to the Practice Excercises:
https://www.db-book.com/Practice-Exercises/index-solu.html

The End