Deadlocks

☐ Consider the partial schedule

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)	

☐ Neither T_3 nor T_4 can make progress — executing **lock-S**(B) causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X**(A) causes T_3 to wait for T_4 to release its lock on A.

☐ Such a situation is called a **deadlock**.

   ☐ To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

15.11   ©Silberschatz, Korth and Sudarshan

Deadlocks (Cont.)

☐ Two-phase locking *does not* ensure freedom from deadlocks.

☐ In addition to deadlocks, there is a possibility of **starvation.**

☐ **Starvation** occurs if the concurrency control manager is badly designed. For example:

   ☐ A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.

   ☐ The same transaction is repeatedly rolled back due to deadlocks.

☐ Concurrency control manager can be designed to prevent starvation.

15.12   ©Silberschatz, Korth and Sudarshan

Deadlocks (Cont.)

☐ The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.

☐ When a deadlock occurs there is a possibility of cascading roll-backs.

☐ Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called **strict two-phase locking** -- a transaction must hold all its exclusive locks till it commits/aborts.

☐ **Rigorous two-phase locking** is even stricter. Here, *all* locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

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.

- **Deadlock prevention** protocols ensure that the system will *never* enter into a deadlock state. Some prevention strategies :
  - Require that each transaction locks all its data items before it begins execution (predeclaration).
  - 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.

More Deadlock Prevention Strategies

- Following schemes use transaction timestamps for the sake of deadlock prevention alone.

- **wait-die** scheme — non-preemptive
  - older transaction may wait for younger one to release data item. (older means smaller timestamp) Younger transactions never Younger transactions never wait for older ones; they are rolled back instead.
  - a transaction may die several times before acquiring needed data item

- **wound-wait** scheme — preemptive
  - older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.
  - may be fewer rollbacks than *wait-die* scheme.

Deadlock prevention (Cont.)

- Both in *wait-die* and in *wound-wait* schemes, a rolled back transactions is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided.

- **Timeout-Based Schemes:**
  - a transaction waits for a lock only for a specified amount of time. If the lock has not been granted within that time, the transaction is rolled back and restarted,
  - Thus, deadlocks are not possible
  - simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval.
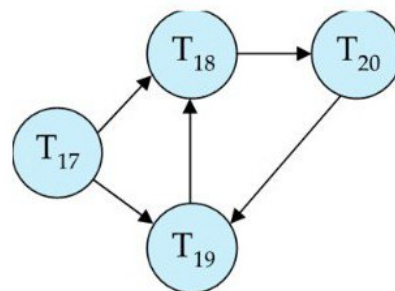
Deadlock Detection

- Deadlocks can be described as a *wait-for* graph, which consists of a pair $G = (V,E)$,

    - V is a set of vertices (all the transactions in the system)

    - E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.

- If $T_i \rightarrow T_j$ is in E, then there is a directed edge from T_i to T_j, implying that T_i is waiting for T_j to release a data item.

- When T_i requests a data item currently being held by T_j, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when T_j is no longer holding a data item needed by T_i.

- The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.

Deadlock Detection (Cont.)
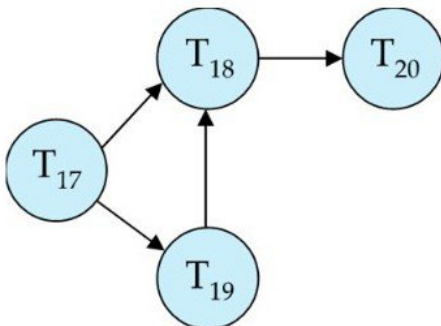
Wait-for graph without a cycle

Wait-for graph with a cycle