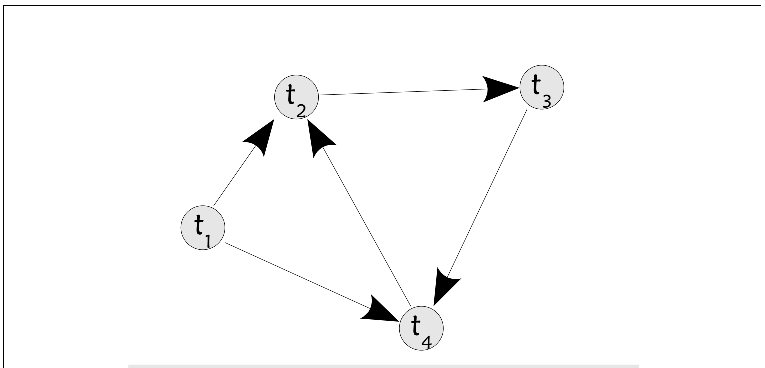
Deadlock

What is a deadlock?
A cycle of transactions after acquiring some locks, waiting for locks to be released by other waiting transactions who themselves have acquired locks is in a state of deadlock. A deadlock is a situation where there is cyclic wait for locks.

Deadlock Detection

- Using Wait-for graph method:-

The graph contains of vertices and directed edges where the vertices are transactions and the arrows are "waiting for" signs.
Example:



Wait-For Graph with Cycle

If there is a cycle existing in the wait-for graph then a deadlock exists.

- Pessimistic Assumption based on time

If a transaction is waiting for too long to acquire a lock, (How to know that the transaction is waiting too long? - Fix a time-out for lock requests.) then it can be presumed that the transaction has entered a deadlock cycle.

Deadlock Prevention / Recovery

There are two approaches to deadlock prevention.

- One approach ensures that no cyclic waits can occur by requiring all locks to be acquired together,
  the disadvantages are:

1. Often hard to predict.
2. Data item utilization is very low. (many will be locked and not used)

- the other approach is **deadlock recovery**.

When a detection algorithm determines that a deadlock exists in the system, the system must takes steps to recover from the deadlock. The most common solution is to rollback one or more transactions. Three actions are required for that.

1. Selection of the Victim transaction (find-out the transaction to abort)

1. What is the progress of the victim
    1. How many locks it has already acquired
    2. How much data it has already used
    3. How many more locks are required for it to proceed
    4. How much more operations are required to finish
2. How many operations are required to rollback

2. Rollback the transaction.
3. Find-out if any **starvation**[$] condition is made with our choice.

Two schemes of deadlock recovery are present (in the point of view of victim selection).

1. Wait and Die Scheme
   Non Preemptive in nature.
   The newer transaction is aborted and again restarted with previous time stamp.
2. Wound – Wait scheme
   Preemptive in nature.
   The older transaction is wounded (aborted) by the younger.

Both wait and die and wound – wait avoid starvation.

In wait and die the a transaction (new) dies simply because the data item which has been requested is now locked. In wound – wait, the new transaction just aborts the older one to control the data.

The major problem with both these schemes is that unnecessary and repeated rollbacks may occur.

Phantom Phenomenon (Just for Short Questions)

When transactions t1 and t2 do not access any tuple in common but still in conflict then they are presumed to be on conflict over a phantom tuple. This problem is called phantom phenomenon[þ].

$ **Starvation**:
When a transaction cannot proceed indefinitely while other transactions are proceeding normally, the transaction is in the state of starvation.
Let's take an example: In a system deadlocks are occurring and a big transaction requiring a large number of locks is always selected to be aborted again and again. As a result the transaction is never able to complete its task and we can call it a case of starvation.
Let's take another example: In a system transaction t1 has got a share level lock, t2 is requesting an exclusive lock on the same item, (A). transaction t3 now requests for a share level lock on (A) and as it can be given because more than one share level lock is allowed on a data object in database, it is given to t3. Serially t4, t5 etc all request shared locks on (A) and they are awarded the lock. Some transactions which has finished their work with (A) releases their locks but (A) is never free for an exclusive lock. So t2 is here in a state of starvation.

þ Let's take two transactions. One computing an aggregate function like sum over a column and another transaction inserting a row. The row is not yet inserted, so the sum calculation will be obsolete as soon as t2 inserts the row. We should not execute t2 before t1 because t2 came later. So there is no common row of conflict among the two transactions. That's why the concept of phantom is brought in.